UNIVERSIDAD DE CANTABRIA

PROGRAMA DE DOCTORADO EN CIENCIA Y TECNOLOGÍA

TESIS DOCTORAL

ENRUTAMIENTO ADAPTATIVO NO MÍNIMO PARA REDES DE INTERCONEXIÓN EFICIENTES

PHD THESIS

NON-MINIMAL ADAPTIVE ROUTING FOR EFFICIENT INTERCONNECTION NETWORKS

Realizada por: **Mariano Benito Hoz**

Dirigida por: **Julio Ramón Beivide Palacio**

Enrique Vallejo Gutiérrez

Escuela de Doctorado de la Universidad de Cantabria

Santander 2020

Non-minimal Adaptive Routing for Efficient Interconnection Networks

Mariano Benito Hoz

supervised by

Dr. Ramón Beivide and Dr. Enrique Vallejo

Doctor of Philosophy

Department of Computer Science and Electronics

University of Cantabria

September 2020

Non-minimal Adaptive Routing for Efficient Interconnection Networks

by **Mariano Benito Hoz**

COLOPHON

This document is typeset using Donald E. Knuth's TEX typesetting system, through `LuaTex`[1] engine by Taco Hoekwater *et al.* implemented in Christian Schenk's `MiKTeX`,[2] and Benito van der Zander's `TeXstudio`[3] editor. Its template is developed over `KOMA-Script srcbook`[4] document class maintained by Markus Kohm. The bibliography is organized with Oliver Kopp's `JabRef`[5] and processed by `BibLaTeX`[6] using `Biber`[7] as its backend, which are mainly maintained by Philip Kime and François Charette respectively. The typeface used for regular text is 12pt Robert Slimbach's Minion Pro.[8] Sans-serif text is written in Slimbach and Carol Twombly's Myriad Pro.[9] Jim Lyles's `Vera Mono`[10] and Claudio Beccari's *Asana Math*[11] fonts are used for `monospaced` and *mathematical* text respectively. Most of the graphics are generated using `gnuplot`.[12]

[1] `http://www.luatex.org`
[2] `https://miktex.org`
[3] `https://www.texstudio.org`
[4] `https://www.ctan.org/pkg/koma-script`
[5] `https://www.jabref.org`
[6] `https://www.ctan.org/pkg/biblatex`
[7] `https://www.ctan.org/pkg/biber`
[8] `https://fonts.adobe.com/fonts/minion`
[9] `https://fonts.adobe.com/fonts/myriad`
[10] `https://www.dafont.com/es/bitstream-vera-mono.font`
[11] `https://www.ctan.org/tex-archive/fonts/Asana-Math`
[12] `http://www.gnuplot.info`

RESUMEN

La humanidad tiene un continuo deseo de evolucionar. Aplicado a la ingeniería de computadores, este ha resultado en la construcción de sistemas computacionales cada vez más rápidos y con más rendimiento. Durante las últimas décadas y acorde a la ley de Moore [162], estos sistemas han duplicado su rendimiento cada dos años. Inicialmente produciendo procesadores tan rápidos como era posible y más recientemente, incrementando el número de *cores*[13] en los microprocesadores [112]. Sin embargo, los requisitos de cómputo de las aplicaciones que cimientan los avances científicos, industriales o sociales exceden en gran medida las capacidades de una única estación de cómputo. Por lo tanto, para dar respuesta a estos requerimientos se agrega la potencia de múltiples nodos de cómputo, dando lugar a lo que habitualmente se conoce como *sistemas de computación paralelos* [12].

Cualquier sistema que emplea múltiples nodos de cómputo para ejecutar una aplicación debe ser diseñado para permitir una comunicación eficiente entre todos ellos. De otra manera, las ventajas del procesamiento paralelo desaparecerían. Por lo tanto, una *red de interconexión* que permita a los diferentes nodos de cómputo compartir información es obligatoria y una pieza clave de los sistemas paralelos. En definitiva, la red de interconexión es el subsistema crítico que convierte un sistema de computación de altas prestaciones en un «supercomputador». Debido a ello y a la constante evolución de estos sistemas, la importancia de las redes de interconexión está en constante crecimiento. Asimismo, estas juegan un rol crucial determinando el rendimiento y el coste total del sistema [59, 54].

El concepto de red de interconexión es definido por Dally y col. [54, p. 2] como «*un sistema programable que transporta información entre terminales*».[14] Es un sistema compuesto por diferentes elementos, principalmente por un conjunto de puntos de intercambio de información llamados *enrutadores* (*routers*) que están interconectados mediante una colección de *cables*. Desde un punto de vista clásico, es programable porque hace las acciones necesarias en sus diferentes elementos para entregar la información transportada desde su origen hasta su destino. La comunicación entre los diferentes nodos de cómputo en una red de interconexión es realizada mediante el intercambio de *mensajes*. Estos son enviados por los enrutadores a través de los múltiples enlaces de red que conectan los diferentes enrutadores entre si. Los mensajes pueden ser divididos en paquetes. Si bien una red de interconexión puede ser usada tanto en el ámbito del sistema como del chip, esta tesis se centra en su aplicación a nivel de sistema, en el cual la red de interconexión permite el intercambio de mensajes entre los diferentes nodos de cómputo que componen un sistema de computación paralelo. Este y otros aspectos introductorios son tratados en el capítulo 1 de manera más exhaustiva. A continuación se presentan las cuatro características principales que definen una red de interconexión:

[13] La Informática es una ciencia reciente y debido a ello, existe un conjunto de términos en inglés que se usan sin traducir en textos en español por no tener estos una traducción clara. Estos son escritos en cursiva.

[14] Texto original: *a programmable system that transports data between terminals*.

- *Topología*: determina el patrón de conexión entre los diferentes enrutadores de la red que define exactamente cómo estos son interconectados mediante cables. La topología es un aspecto crucial de la red de interconexión porque establece sus límites de rendimiento determinando su diámetro y su bisección, así como su diversidad de caminos. El diámetro de una red es el mayor número de saltos entre dos enrutadores usando caminos mínimos. La bisección de la red es el menor conjunto de enlaces que divide la red en dos partes iguales. La diversidad de caminos determina el número de caminos mínimos existentes entre la mayoría de parejas de nodos de cómputo. El trabajo de doctorado mostrado en esta tesis basa su investigación principalmente en la topología Dragonfly [108], concretamente en su variante conocida como canónica y con un patrón de conexionado de los cables globales denominado *palmtree* [38].

- *Enrutamiento*: determina qué camino recorre un mensaje entre su nodo de cómputo origen y destino. El algoritmo de enrutamiento determina la fracción de rendimiento que es obtenida del límite impuesto por la topología y balancea la carga de la red bajo patrones de tráfico adversos. Esto último se puede realizar aplicando directamente el algoritmo de balanceo de carga propuesto por Valiant (VLB, [187, 186]), el cual consiste en enviar cada paquete a un nodo aleatorio de la red (R_{ROOT}) y después enviar dicho paquete desde ese nodo a su verdadero destino. Si además se desea enviar el tráfico por rutas mínimas cuando sea posible, se debe contemplar el uso de algoritmos de enrutamiento adaptativos que seleccionan entre rutas mínimas o no mínimas basándose en el estado de la red. En una red Dragonfly, estos algoritmos están principalmente representados por UGAL [172] y *Piggyback* (PB, [96]). UGAL utiliza la información de congestión en los puertos asociados a las rutas mínima y no mínima para decidir cuál emplear, lo que implica que necesita que la congestión se propague hasta el enrutador origen para detectarla. Cuando la congestión es remota, lo cual ocurre típicamente en la Dragonfly, PB aporta sobre UGAL información compartida entre diversos enrutadores.

- *Control de flujo*: reserva los diferentes recursos de la red a los paquetes que la están atravesando y puede detectar y prevenir situaciones de *deadlock* [180] y *livelock* [59]. Este aspecto adquiere relevancia cuando la utilización de los recursos es elevada. Los mecanismos de control de flujo emplean cierta información para notificar a otros enrutadores su capacidad para recibir mensajes. Esta información puede indicar la disponibilidad como los mensajes de *pausa* en *Ethernet* o la cantidad de espacio libre como los *créditos* de Infiniband. Habitualmente, estos mecanismos se aplican salto a salto pero también hay técnicas similares para manejar la congestión punto a punto como las *notificaciones explícitas de congestión* (*Explicit Congestion Notification*, ECN).

- *Arquitectura del enrutador*: define la organización del enrutador incluyendo el *switch fabric*, los *buffers* y la lógica de control asociada a estos elementos. La arquitectura del enrutador está muy relacionada con el enrutamiento y el control de flujo que va

a ser implementado en la red. Los enrutadores pueden exponer ciertos contadores a disposición de los administradores de red y de su propia arquitectura. Los enrutadores pueden ser particionados de manera lógica en tres planos o capas: el plano de datos que realiza la transferencia de un paquete desde una interfaz a otra del enrutador, el plano de control que determina cómo deben ser transferidos los paquetes y el plano de gestión que permite la configuración y monitorización del enrutador.

Cada uno de los aspectos mencionados anteriormente determinan como la red de interconexión es implementada y establecen un límite a su rendimiento y coste. El rendimiento de una red de interconexión es cuantificado principalmente mediante dos métricas: el *ancho de banda* (*throughput*) y la *latencia* (*latency*). El ancho de banda indica la tasa de transferencia de datos mantenida que se puede conseguir en la red y la latencia el tiempo transcurrido en trasferir un mensaje desde su origen a su destino. Una cota clave del rendimiento es el punto de saturación, en el cual el ancho de banda llega a su pico máximo por lo que ya no puede soportar más aunque se incremente la carga ofrecida a la red. Asimismo, otro factor estudiado cuando se analiza una red de interconexión es la *equidad* (*fairness*) asignando los recursos a los diferentes nodos de cómputo.

La evolución de la tecnología ha propiciado la convergencia entre los sistemas de supercomputación (*High-Performance Computing*, HPC) y los sistemas de centros de datos empresariales (*Data Center*, DC). Los sistemas para HPC son desarrollados acorde a los requisitos de las aplicaciones de alta computación, las cuales habitualmente son intensivas en comunicación y sensibles a la latencia de red. Los sistemas para DC empresariales habitualmente se orientan a proveer servicios a través de internet, como páginas web o correo electrónico, y estos no son muy sensibles a la latencia, dado que operan en las escalas de tiempo de las personas que los utilizan. Sin embargo, las aplicaciones de *back-end* que proveen estos servicios pueden requerir una comunicación intensiva y baja latencia. Esta tesis se centra en sistemas HPC, aunque sus contribuciones pueden ser aplicadas a las redes de grandes DCs. Tradicionalmente, la tecnología Ethernet «*commodity*» ha sido empleada en entornos domésticos y DC empresariales. Sin embargo, la tecnología Ethernet está adquiriendo una relevancia considerable en los sistemas HPC. En la última lista Top500 [179], que registra los 500 supercomputadores más veloces del momento, dicha tecnología representa una porción ligeramente superior a la mitad de los sistemas.

Recordando el concepto de programabilidad y siguiendo la tendencia marcada por el desarrollo de la computación en la nube, se introdujo el concepto de las redes definidas por software (*Software-Defined Networking*, SDN). Esta propuesta aspira a hacer las redes ágiles y flexibles, lo que permite que sean inteligentes y estén controladas de manera centralizada usando aplicaciones. SDN sugiere el uso de un controlador centralizado que directamente controle los múltiples dispositivos de enrutamiento presentes en la red. Además, el concepto ha continuado evolucionando y su siguiente paso ha sido permitir la definición del comportamiento de los dispositivos de red mediante un lenguaje de configuración independiente del dispositivo mientras que mantiene centralizado el control de los mismos. Estos dispositi-

vos de red son conocidos como dispositivos PISA (*Protocol Independent Switch Architecture*) y permiten que el plano de datos sea programado. El capítulo 2 analiza en profundidad los aspectos mencionados anteriormente y otros temas fundamentales de las redes de interconexión que son relevantes para esta tesis.

Como se ha visto, el primer aspecto a especificar para definir una red de interconexión es su topología, la cual está muy influenciada por las restricciones de colocación de los diferentes elementos del sistema. Las redes escalables de bajo diámetro que son eficientes en potencia y coste, como la Dragonfly, están basadas en topologías que tratan de acercarse al límite de Moore[15] [82, 135]. Estas redes sacrifican la diversidad de caminos para obtener la máxima escalabilidad, lo que favorece que sus enlaces se congestionen en el momento que el tráfico deja de ser benigno. Por lo tanto, una vez definida la topología y por ende los límites de rendimiento de la red, se debe diseñar un algoritmo de enrutamiento que se acerque lo máximo posible a estos límites y, debido a la ausencia de diversidad de caminos mínimos, dicho enrutamiento debe explotar los caminos no mínimos cuando el patrón de tráfico es adverso. Típicamente, estos algoritmos de enrutamiento se implementan de manera adaptativa mediante la comparación de los créditos disponibles en los diferentes puertos de salida. Los caminos no mínimos habitualmente son construidos de acuerdo a la propuesta de Valiant que implica rutas no mínimas que doblan la longitud de las rutas mínimas y por lo tanto, doblan la latencia media soportada por los paquetes.

El trabajo de doctorado expuesto en esta tesis tiene como objetivo el diseño de algoritmos de enrutamiento adaptativo no mínimo para redes de interconexión eficientes. Este objetivo es abordado dividiéndolo en dos subobjetivos ortogonales. El primero es motivado por el proyecto Mont-Blanc [157], cuyo objetivo es realizar un sistema HPC basado en sistemas en chip (*Systems on a Chip*, SoCs) de ARM diseñados inicialmente para dispositivos móviles. Estos SoCs habitualmente tienen una interfaz de red Ethernet y, debido a la ausencia de créditos en dicha tecnología, surge el propósito de diseñar un enrutamiento sobre esta tecnología capaz de seleccionar entre rutas mínimas y no mínimas sin basarse en créditos. El segundo subobjetivo es motivado por la existencia de múltiples caminos no mínimos entre los enrutadores en las topologías de bajo diámetro [54] como la Flattened Butterfly [110] o la Dragonfly [108] y la posibilidad de adaptar el enrutamiento a las condiciones de la red [54, 110, 108]. Entonces, surge la meta de adaptar el número de saltos en las rutas no mínimas a las condiciones de la red, en vez de utilizar directamente rutas acordes al algoritmo de balanceo de carga de Valiant. Durante el periodo formativo del doctorado han sido exploradas diversas ideas con la finalidad de conseguir los objetivos mencionados anteriormente. Algunas de ellas, expuestas en esta tesis, han dado lugar a diferentes propuestas que han sido publicadas en foros relevantes para el área [23, 24, 21, 25, 22]. Otras publicaciones relacionadas han sido desarrolladas durante el mismo periodo [157, 67].

[15] El límite de Moore indica el máximo número de nodos que un grafo puede albergar dado el grado de un nodo y el diámetro del grafo.

Las propuestas presentadas en esta tesis se comparan con otras actuales mediante las tres métricas introducidas previamente: ancho de banda, latencia y equidad. Estas métricas se obtienen mediante simulación dado que son introducidas en un nivel inicial de su desarrollo. Para evaluarlas se emplea principalmente el simulador de red FOGSim [73] puesto que ofrece la precisión necesaria para evaluarlas y sus requisitos computacionales, así como el tiempo requerido para los experimentos, son asumibles. Para evaluar las propuestas se diseñan diferentes experimentos que consisten en someter a la red simulada a una serie de cargas para evaluar el rendimiento que ofrece el algoritmo de enrutamiento estudiado. Los experimentos son llevados a cabo mediante cargas sintéticas que exponen a la red evaluada a diferentes escenarios. Estas cargas representan estados estacionarios con diferentes patrones de tráfico aislados o mezclados y cargas transitorias que varían el patrón de tráfico en diferentes momentos del tiempo, lo cual sirve para evaluar la respuesta de las propuestas a dichos cambios. Los experimentos se realizan una batería de veces y se promedia el resultado de los mismos. El capítulo 3 amplía esta información y ofrece todos los detalles acerca de la metodología empleada durante el desarrollo de esta tesis.

El capítulo 4 introduce una implementación realista y competitiva de una red Ethernet escalable y sin pérdidas para entornos HPC, que fue publicada en [23], y para la que se han considerado topologías de bajo diámetro enfocadas a reducir el gasto energético. La arquitectura propuesta se basa en: un direccionamiento jerárquico, técnicas de compactado de las tablas de enrutamiento, un mecanismo dinámico para mantener las direcciones jerárquicas de los nodos y unas reglas de enrutamiento condicionales que son instanciadas en los enrutadores de manera proactiva. La implementación de los mecanismos mencionados es realista y requiere cambios mínimos sobre un dispositivo que soporte SDN, lo que permite desplegar redes de baja potencia y bajo diámetro para sistemas HPC basados en tecnología Ethernet.

La mayoría de los enrutamientos adaptativos propuestos para redes de bajo diámetro se basan en indicadores locales de congestión, como los créditos, para decidir cómo enrutar los paquetes. Sin embargo, este capítulo, debido a la ausencia de créditos en Ethernet, propone dos enrutamientos adaptativos: *MAR-bP* publicado en [23] y *QCN-Switch* en [24, 25]. Respectivamente, estos se adaptan al estado de la red basándose en los mensajes de *pausa* de Ethernet o mensajes ECN.

La arquitectura de red introducida obtiene un ahorro energético de hasta un 54% sobre una red Ethernet sin las modificaciones propuestas. El rendimiento de MAR-bP presenta ciertas limitaciones que han sido superadas por QCN-Switch. El rendimiento de QCN-Switch en estado estacionario es comparable al de otros enrutamientos sofisticados propuestos para HPC, como PB. Considerando tráficos transitorios, QCN-Switch no se adapta tan rápido como los enrutamientos basados en créditos, no obstante, responde en menos de un milisegundo, lo que habitualmente es suficiente para la mayoría de las aplicaciones. Un análisis de sensibilidad a los parámetros analiza las elecciones asumidas durante el diseño de QCN-Switch.

El algoritmo de Valiant usado para balancear la carga entre los diferentes enlaces de la red obtiene un buen rendimiento sobre patrones de tráfico adversos en redes de bajo diámetro como la Dragonfly. Sin embargo, este dobla la longitud de la ruta no mínima e incrementa la latencia base. El capítulo 5, que dio lugar a las publicaciones [21, 22], introduce dos mejoras sobre VLB: 1) *RVLB*: un mecanismo VLB restringido que selecciona el nodo R_{ROOT} en la misma partición en la que se encuentran el enrutador origen y destino para mejorar el rendimiento en tráficos con localidad, y 2) *RVLB-Recomp*: un mecanismo de recómputo que incrementa la inyección a la red seleccionando nodos R_{ROOT} alternativos cuando el puerto seleccionado en el ciclo de enrutamiento previo está bloqueado.

Basándose en esos dos mecanismos, el capítulo 5 también introduce el enrutamiento *ACOR: Adaptive Congestion-Oblivious Routing*. El objetivo de ACOR es mejorar la latencia del caso común reduciendo el número de saltos en la ruta no mínima, mientras que soporta patrones de tráfico patológicos con rutas no mínimas más largas. ACOR aplica RVLB no solo a tráficos con cierta localidad sino a todo el tráfico y extiende la idea del recómputo para adaptar el enrutamiento a las condiciones de la red. Esto se realiza cambiando la ruta no mínima según una secuencia de políticas ordenadas por la longitud de la ruta que generan. ACOR previene la variabilidad en los resultados mediante un mecanismo de histéresis con una implementación relativamente simple.

De la misma forma que VLB, ACOR no envía tráfico de manera mínima, por lo que su rendimiento en tráficos benignos no es óptimo. Por esta razón, ACOR es emparejado con un enrutamiento que selecciona entre rutas mínimas y las rutas no mínimas propuestas que propone para cada paquete inyectado. La combinación de PB[16] y ACOR da lugar a *PB-ACOR*, el cual selecciona la ruta no mínima más corta posible solamente cuando la ruta mínima se encuentra congestionada. Este mecanismo mantiene los beneficios de ACOR para tráficos adversos y es competitivo en tráficos benignos.

Los resultados de la evaluación presentada en el capítulo 5 muestran que todas las variantes de ACOR evitan cualquier limitación del rendimiento derivada de usar rutas con una longitud inadecuada. Además proporciona equidad en la asignación del ancho de banda entre los diferentes terminales y reduce la latencia base hasta en un 28% comparado con una modificación de VLB que tiene aplicadas las dos mejoras propuestas (RVLB y RVLB-Recomp). Además, en tráficos benignos PB-ACOR obtiene un buen ancho de banda y su latencia es óptima y en tráficos adversos obtiene un ancho de banda competitivo y una reducción en latencia significativa comparado con el enrutamiento PB.

Siguiendo la tendencia de ACOR, el capítulo 6 propone el algoritmo de enrutamiento *LIAN: Latency-Improved Adaptive Non-minimal routing for Dragonfly Networks*, el cual ha sido enviado recientemente a una revista y se encuentra en proceso de revisión. Basándose en las condiciones de la red, LIAN minimiza el número de saltos en las rutas no mínimas

[16] Se ha usado el enrutamiento PB porque ACOR no puede ser implementado directamente sobre la arquitectura de red introducida en el capítulo 4 debido a restricciones tecnológicas. Sin embargo, la combinación de QCN-Switch y ACOR podría plantearse sobre dispositivos de red PISA.

siempre que esto no suponga un detrimento del rendimiento. Para inferir el patrón de tráfico presente en la red y determinar las condiciones de la red, LIAN emplea unos contadores *extendidos* que representan la carga ofrecida. Estos contadores propuestos son comparables a los que ya ofrecen los enrutadores para entornos HPC [49, 131]. Los contadores *global* propuestos representan la carga ofrecida a cada grupo y basándose en estos, los contadores *intermediate-local* son derivados mediante un simple cálculo combinando algunos de los primeros. La ecuación que controla cómo deben combinarse los contadores para extraer información de ellos es consecuencia de la simetría ofrecida por la disposición de los cables globales en la topologia empleada.

Los contadores empleados son usados para inferir la congestión en los grupos intermedios y analizar la demanda de los puertos globales. Basado en ello, LIAN es capaz de determinar la longitud de la ruta no mínima que deben seguir los paquetes. Esta decisión se realiza paquete a paquete y dinámicamente se minimiza la longitud de los caminos no mínimos. LIAN supera la necesidad de ACOR de analizar la congestión en el enrutador fuente porque está basado en inferir la carga del tráfico en vez de en el bloqueo de los puertos del enrutador. Por lo tanto, LIAN no necesita hacer uso de PB y compartir información. Para seleccionar entre rutas mínimas y no mínimas, LIAN modula los parámetros del algoritmo UGAL subyacente basándose en los contadores globales.

Los resultados de la evaluacion de LIAN exponen que los contadores extendidos correctamente infieren el patrón de tráfico en todos los posibles rangos de carga, proporcionando un enrutamiento adecuado bajo tráficos benignos y adversos. LIAN obtiene un rendimiento igual o mejor que otros enrutamientos seleccionando rutas no mínimas con el número de saltos ideal para no incurrir en congestión y sin dar más saltos de los necesarios. Esto consigue una reducción en latencia de hasta un 30.0% sobre algoritmos de enrutamiento actuales propuestos para sistemas HPC. Además, el rendimiento que obtiene es muy cercano al límite teórico que presenta la topología para cada patrón de tráfico analizado, lo que convierte a LIAN en un enrutamiento muy competitivo. Tampoco se han apreciado signos de *unfairness* y el rendimiento que obtiene es estable pasado el punto de saturación. Además, el tiempo de reacción de LIAN a cambios de tráfico es cuasi-inmediato, tanto para cambios entre diferentes patrones de tráfico como para cambios entre diferentes valores de carga sobre el mismo patrón. En conclusión y hasta donde se conoce, LIAN es el primer enrutamiento adaptativo en el enrutador origen que combina las propiedades deseables de: 1) basarse sólamente en información local al enrutador, 2) emplear la ruta no mínima más corta dentro de lo que permiten las condiciones de la red, y 3) proporcionar un rendimiento estable.

El trabajo de doctorado reflejado en esta tesis surge con el objetivo de diseñar un enrutamiento adaptativo no mínimo para redes de interconexión eficientes. El cual se divide en dos subobjetivos ortogonales: 1) diseñar un enrutamiento capaz de seleccionar entre rutas mínimas y no mínimas sin emplear créditos, y 2) diseñar un enrutamiento capaz de adaptar el número de saltos en las rutas no mínimas a las condiciones de la red en vez de usar directamente rutas no mínimas según VLB. En base a lo expuesto anteriormente, se puede concluir

que los dos subobjetivos han sido satisfechos porque el capítulo 4 cubre el primer subobjetivo proporcionando el algoritmo de enrutamiento *QCN-Switch*, el cual decide si enviar cada paquete por una ruta mínima o no mínima basándose en mensajes ECN capturados de la red y los capítulos 5 y 6 proporcionan dos algoritmos de enrutamiento diferentes para optimizar el número de saltos en las rutas no mínimas. Estos además realizan la selección entre rutas mínimas y no mínimas basándose en PB o UGAL y en créditos. Por esta razón, una combinación entre LIAN y QCN-Switch es emplazado directamente como trabajo futuro.

El resultado de esta tesis de doctorado es un conjunto de propuestas que permiten implementar un enrutamiento adaptativo no mínimo para redes de interconexión eficientes y que contribuyen a esto desde dos puntos de vista ortogonales. Por un lado, la propuesta *QCN-Switch* implementada sobre la arquitectura presentada permite desplegar redes de interconexión escalables para entornos HPC sobre dispositivos de red Ethernet empleando un routing adaptativo con un rendimiento que rivaliza frente a otras propuestas de enrutamiento diseñadas específicamente para HPC. Por otro lado, *ACOR* permite hacer ingeniería de tráfico sin tener en cuenta la topología y adaptar la latencia de los paquetes a las condiciones actuales de la red. Adicionalmente, *LIAN* mejora la propuesta previa y minimiza la longitud de las rutas no mínimas en una topología Dragonfly con despliegue Palmtree.

Esta tesis ha desarrollado ambos sub-objetivos marcados y con ello resuelto el objetivo general. Para ello se han hecho diversas contribuciones, sin embargo, cada propuesta abre la puerta a nuevos retos, los cuales pueden ser agrupados como nuevas líneas de investigación. Algunas de ellas pueden ser construidas sobre el trabajo mostrado en esta tesis y son listadas a continuación:

- *Implementar y desarrollar* LIAN *junto con la propuesta* QCN-Switch *sobre dispositivos de red compatibles:* Implementar LIAN sobre QCN-Switch usando dispositivos PISA para proporcionar una red escalable basada en la topología Dragonfly y construida sobre dispositivos de red Ethernet, prestando atención al ahorro energético y optimizando la latencia media sufrida por los paquetes.

- *Identificar y manejar por separado la congestión en los terminales y la congestión en la red:* En el caso de que un terminal de la red no pueda aceptar más tráfico, explotar la diversidad de caminos mediante enrutamientos adaptativos no mínimos incrementa la presión en la red de interconexión y puede reducir el rendimiento en vez de incrementarlo. La capacidad de detectar estas situaciones combinada con algún mecanismo de restricción de la inyección puede prevenir la propagación de la congestión a toda la red y la degradación del rendimiento.

- *Colaboración entre los terminales y los dispositivos de red:* Trasladar parte de las decisiones de enrutamiento a los nodos de cómputo, teniendo en cuenta que estos pueden obtener información del estado de la red mediante los enrutadores. Además, se puede aprovechar la capacidad de computación de los terminales para tomar mejores deci-

siones de enrutamiento y liberar de parte de estas tareas a los dispositivos de red, con lo que se podría mejorar la eficiencia interna de los mismos.

- *Trasladar* RVLB *a otras topologías:* Identificar la localidad en una red Dragonfly o Flattened Butterfly es trivial, ya que ambas definen un concepto en el que basarla, como es respectivamente el grupo o la dimensión. Sin embargo, cómo aplicar RVLB a otras topologías sin sufrir congestión bajo cualquier patrón de tráfico no es trivial.

- *Trasladar los contadores de* LIAN *dependientes de la topología a otras topologías:* La simetría rotacional explotada en el capítulo 6 depende de la topología Dragonfly y del conexionado global Palmtree para optimizar el enrutamiento mediante los contadores *Intermediate-local* y no es implementable directamente en otras topologías o en otras disposiciones de los cables globales en una red Dragonfly. Sin embargo, este estudio puede ser interesante.

- *Implementar y desarrollar* ACOR *sobre dispositivos PISA:* La aparición de dispositivos de red en los que se puede programar el camino de datos podría permitir la implementación del mecanismo de re-cómputo de rutas y con esto, el enrutamiento ACOR propuesto podría ser implementado. La selección de la ruta no mínima que hace ACOR puede ser combinada con PB o UGAL, los cuales están basados en créditos, o sobre la arquitectura propuesta para entornos HPC, la cual basa esta decisión en mensajes de congestión explícitos como los implementados por Ethernet 802.1Qau.

- *Emplear los contadores introducidos en* LIAN *también para decidir si encaminar cada paquete siguiendo una ruta mínima o no mínima:* Los contadores de tráfico empleados para modular UGAL y determinar la longitud de la ruta no mínima podrían ser usados para determinar directamente si enrutar los paquetes por una ruta mínima o no mínima y con ello prescindir de UGAL y de los créditos.

- *Extender* LIAN *para determinar la longitud de la ruta no mínima completa:* LIAN optimiza la primera parte de la ruta no mínima basándose en las condiciones de la red detectadas pero no analiza la segunda parte que se forma mediante una ruta mínima entre el nodo intermedio R_{ROOT} y el destino real del paquete. La selección del nodo intermedio puede ser restringida para determinar completamente la longitud de la ruta desde el enrutador origen.

AGRADECIMIENTOS - *ACKNOWLEDGMENTS*

Esta tesis doctoral representa el trabajo de seis años y ahora que tocan a su fin, se me antojan como un largo *rally*. Para los profanos en el tema, en un *rally* se disputan una serie de tramos cronometrados. Entre una consecución de tramos y la siguiente, se reagrupa a los equipos que continúan participando y se realiza una asistencia en la que se arregla cualquier desperfecto. Los tramos discurren por carreteras cerradas al tráfico y de las que, habitualmente, merece la pena disfrutar, tanto de la conducción como de los paisajes. Muchos han sido los contemplados desde mi *baquet*, algunos tan bellos que hasta consiguieron distraerme de mi labor de «canta notas», muchas las personas que he ido conociendo en cada reagrupamiento y muchos los momentos vividos junto a ellas, algunos tan inolvidables que no quería dejar pasar este *refuelling* sin hacerles llegar mi gratitud. Abusando de la terminología, ahora estoy acercándome al parque cerrado de fin de *rally* pero no por ello he olvidado el momento en el que me inscribí, ni lo vivido durante esta etapa tan especial para mí. Es por ello, que me gustaría empezar por dar las gracias a mis directores de tesis, el Dr. Ramón Beivide y el Dr. Enrique Vallejo.

Mon y Enrique, Enrique y Mon, ambos habéis ido conmigo en el coche y me habéis aportado todo vuestro apoyo y dedicación para que concluyese este trabajo con la mayor satisfacción posible. Es por esto, que querría agradeceros todo vuestro esfuerzo y confianza depositada en mí. En particular, Mon, muchas gracias por abrirme la puerta del grupo (en múltiples ocasiones), por orientarme y por compartir innumerables momentos en los que discutir ideas y charlar abiertamente de lo que en ese momento tuviésemos entre manos, ya fuese profesional o personal. Enrique, muchas gracias por haber trabajado constantemente a mi lado, por tu cercanía, profesionalidad y rigurosidad, así como por haberme permitido llegar hasta aquí. Mis gracias más sinceras a ambos por haberme acompañado por los tramos disfrutados durante estos seis años.

Me gustaría dar las gracias a todos los miembros del grupo de Arquitectura y Tecnología de Computadores. Empezando por los séniors, muchas gracias por ser como «la organización», ya que sin vosotros no habría *rally*. Todo vuestro trabajo y dedicación es lo que nos permite a los juniors estar investigando a vuestro lado y especialmente a J. Luis, me gustaría agradecerte la confianza mostrada en mí desde que estaba a las puertas de acabar la ingeniería y también por haberme dado la salida en tramos que al final no disputé, es por esto último que también querría pedirte disculpas. Continuando con los júniors, me gustaría agradeceros que habéis estado siempre disponibles para aportar vuestra ayuda y especialmente a Iván, muchas gracias por las interminables horas compartiendo despacho y conversaciones variopintas en las cuales tan pronto nos tocaba un tramo de tierra como uno de asfalto. Raúl, gracias por unirte a nosotros cuando aparecías por el despacho como si siempre estuvieses allí. Pablo, gracias por haberme abierto los ojos a los entresijos de ese hijo putativo que tenemos a medias llamado FOGSim.

Querría dar las gracias a «mi familia», que al contrario de la definición estándar, esta es para mí un órgano vivo que evoluciona a lo largo de los años y no se ciñe a personas

emparentadas sino que abarca a las personas que aunque parece que van y vienen, siempre permanecen a nuestro lado. Vuestro apoyo incondicional siempre que ha sido necesario, a veces en la cercanía y a veces en la lejanía, es lo que me ha impulsado tramo a tramo y me ha permitido terminar este *rally*.

Me gustaría dar las gracias a Roberto. Me dirigiste en mi anterior *rally* y me permitiste comenzar esta carrera de fondo. Muchas gracias por haberme puesto tan fácil la decisión de «abandonar el barco» y a la vez por haber seguido a mi lado pendiente de mi progreso. Igualmente y siguiéndome más de cerca, gracias a mis compañeros de «Benito & Coria's food» por haber aguantado la explicación del tramo en el que me encontraba en cada una de nuestras comidas.

I would like to thank all the colleagues of Recore Systems B.V. You guys were part of one of the special stages of this rally and thank you for accompanying me during my stay in The Netherlands. Specially, thank you so much, Sharan and Michiel for your cheerful long days of work, specially the after-hours. Your devotion for the work was an inspiration to accomplish my internship project and key for my incorporation to Recore.

Por último y no por ello menos importante, me gustaría expresar mi agradecimiento más sincero a mi abuela y a mi pareja. Güeli, muchas gracias por haber desencadenado mi inscripción al doctorado y por haberme guiado hasta aquí aun sin estar presente, sé que estarías muy orgullosa de este momento. Ahora que estoy llegando al parque cerrado de este y como en casi todos los *rallys* en los que participo, no puede faltar una foto en el pódium con Sandra. Muchas gracias, cariño, por tantas razones que me permitirían reescribir todas y cada una de las páginas de esta disertación, aunque especialmente, por haber estado en todas las asistencias de este *rally* y por haber sido mi apoyo en innumerables momentos. Y quiero decirte también que justamente eso, inolvidables momentos, son los que nos quedan por vivir, por lo que el próximo *rally* es nuestra vida juntos.

I would like to thank the anonymous reviewers of the papers which support this PhD thesis, for their valuable comments and suggestions to improve the quality of them and so, for indirectly improving the quality of part of the work presented in this dissertation.

This work has been supported by the Spanish Ministry of Education, Culture and Sports under grant FPU14/02253, the Spanish Ministry of Economy, Industry and Competitiveness under contracts TIN2010-21291-C02-02, TIN2013-46957-C2-2-P, and TIN2016-76635-C2-2-R (AEI/FEDER, UE), the Spanish Research Agency under contract PID2019-105660RB-C22/AEI/10.13039/501100011033, the European Union under agreements FP7-ICT-2011-7-288777 (Mont-Blanc 1) and FP7-ICT-2013-10-610402 (Mont-Blanc 2), the University of Cantabria under project PAR.30.P072.64004, and by the European HiPEAC Network of Excellence through an internship grant supported by the European Union's Horizon 2020 research and innovation program under grant agreement No. H2020-ICT-2015-687689.

Thank you very much to you all!

To my beloved Sandra.

ABSTRACT

Interconnection network is a key concept of any parallel computing system. Since two or more components of a digital system are connected, an interconnection network which allows these components to share data is required. The first aspect to define an interconnection network is its topology, which is tied to system packaging constraints. Typically, power and cost-efficient scalable networks with low diameter rely on topologies that approach the Moore bound in which, in order to get the maximum size, there is no minimal path diversity. Once the topology is defined, the performance bounds of the network are determined consequently, so a suitable routing algorithm should be designed to accomplish as much as possible of those limits and, due to the lack of minimal path diversity, it must exploit non-minimal paths when the traffic pattern is adversarial. These routing algorithms usually select between minimal and non-minimal paths based on the network conditions, where the non-minimal paths are built according to Valiant load-balancing algorithm. This implies that these paths double the length of minimal ones and then the latency supported by packets increases. Regarding the technology, from its introduction in HPC systems in the early 2000s, Ethernet has been used in a significant fraction of the systems. Moreover, Ethernet's large economy of scale, the advent of simple white-box switches, the possibility of lossless implementations, and the ubiquity of Ethernet NICs suggest that it will remain as a cost-effective alternative for HPC interconnection networks.

This work pursues the design of non-minimal adaptive routing algorithms for efficient interconnection networks, dividing it in two sub-objectives. The first is a routing capable of selecting between minimal and non-minimal paths without relying on flow-control credits, which are not available in commodity Ethernet technology. The second is a routing capable of employing variable length non-minimal paths depending on the network conditions to improve average latency.

This dissertation introduces a realistic and competitive implementation of a scalable lossless Ethernet network for HPC environments considering low-diameter and low-power topologies. This allows for up to 54% power savings. Furthermore, it proposes a routing upon the cited architecture, hereon *QCN-Switch*, which selects between minimal and non-minimal paths per packet based on explicit congestion notifications instead of credits. Once the miss-routing decision is implemented, it introduces two mechanisms regarding the selection of the intermediate switch to develop a source adaptive routing algorithm capable of adapting the number of hops in the non-minimal paths. This routing, hereon *ACOR*, is topology-agnostic and improves average latency in all cases up to 28%. Finally, a topology-dependent routing, hereon *LIAN*, is introduced to optimize the number of hops in the non-minimal paths based on the network live conditions. Evaluations show that LIAN obtains almost-optimal latency and outperforms state-of-the-art adaptive routing algorithms, reducing latency by up to 30.0% and providing stable throughput and fairness.

Contents

List of Figures

List of Tables

1 Introduction

Humanity has a lot of qualities and one big and continuous desire of evolving. When applied to computing engineering, this has led to the construction of faster and more capable computing systems each time. During the last decades and according to Moore's law [162], it has been able to duplicate the performance of the systems every two years, firstly producing microprocessors as fast as possible and more recently, increasing the number of cores inside the microprocessors [112]. Nevertheless, the computational requirements of *Data Center* (DC) high-demanding applications and *High-Performance Computing* (HPC) Systems, which are the foundation for scientific, industrial and societal advancements, far exceed the capabilities of a single computing host. Hence, it is usually needed to aggregate the power of several terminals to confront these computational requirements. This type of machines are denoted as *parallel* [12] computing systems.

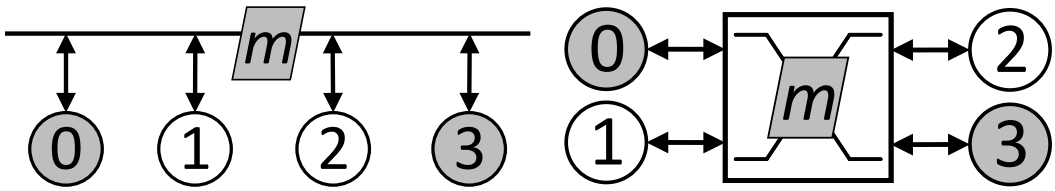
Any parallel system that employs multiple computing hosts to run an application must be designed to allow an efficient communication between these computing nodes; otherwise, the advantages of parallel processing may disappear due to an inefficient communication. Hence, an *interconnection network* that allows the joined computing systems to share data is mandatory and a key part of those parallel systems. In other words, the interconnection network is the critical subsystem of an HPC system that makes it a "supercomputer".

This chapter extends the previous paragraphs introducing and remaking the importance of interconnection networks in Section 1.1. Section 1.2 presents the objective of the PhD work concluded with this dissertation, and exposes how it is approached. The major contributions of this work are described in Section 1.3. Finally, this introductory chapter is concluded in Section 1.4, which presents how the remainder of this dissertation is structured.

Chapter contents

1.1 Interconnection networks

The interconnection network is a key concept of any parallel computing system and it can be defined as "*a programmable system that transports data between terminals*," according to Dally *et al.* [54, p. 2]. Since two or more components of a digital system are connected, an interconnection network, which allows these components to share data at the form of *messages*, is required. It is a system made up of different parts, mainly by a set of switching elements interconnected by a collection of cables. As a classical view, the network is programmable[1] in the sense that its makes the necessary actions on its different elements to deliver the transported messages from their sources to their destinations. Figure 1-1 illustrates the functional view of two interconnection networks with four terminals connected to the network through bidirectional network links and a message *m* exchanged between terminals zero and three. To simplify the network diagrams and since all links are bidirectional, the interconnection links in the diagrams are hereon represented with a simple line, i.e., without arrows.



(a) Bus (b) Crossbar

Figure 1-1: Conceptual representation of (a) bus and (b) crossbar interconnection networks interchanging a message *m* between terminals 0 and 3. All interconnection links are bidirectional.

Interconnection networks are omnipresent nowadays, and they are used in almost all digital systems that have two or more components to be connected. They are widely used in computer systems at multiple points, from on-chip networks up to interconnecting computing hosts at *system level*. HPC architectures are seeing a large growth in size and complexity in an attempt to achieve the performance levels required by exascale computing [183]. As the number and complexity of components in a system continues to increase, the impact of the interconnection network on the overall system performance and cost also increases. Nowadays, the performance of most systems in this field is limited by their communication, not by their logic or memory. In a multiprocessor computer system, as processor and memory performance continues increasing according to Moore's law, the performance of the interconnection network plays an important role to determine the overall performance of the system, as well as its cost [59, 54].

Communication between the different terminals (also called *computing nodes*, *compute hosts* or *endpoints*) in an interconnection network is performed by sending *messages*. These

[1] This concept is revisited later to modernize the definition of the adjective programmable.

are sent by the switching nodes of the network or *routers* through network links or *channels*. Messages may be broken into one or more *packets*, which is the smallest unit of information that contains routing and sequencing information, for its transmission. In turn, a packet contains one or more flow control units or *flits*, where a flit is the smallest unit on which flow control is performed. Data information is transferred over physical links in physical transfer units or *phits*. Commonly, the exchange of information between a source and a destination pair is denoted as a *flow*.

An interconnection network can be used mainly at two levels; *system-level* and *on-chip*. At the former, interconnection networks allow the exchange of messages between the multiple computing hosts which compound a parallel computing system and at the latter, they interconnect the several elements which compound the chip, for example, multiple cores and memories in a multiprocessor.

A bus network, like the presented in Figure 1-1(a), is a very simple interconnection network and it is not capable of transferring more than one piece of information at a given time between any two terminals due to the sharing of the cable connecting all of them. Moreover, as that network does not scale and as the number of terminals increases, the obtained bandwidth between them is drastically reduced. On the other hand, using a crossbar, like the network represented in Figure 1-1(b), in which all the terminals are connected to each other in a point-to-point way, all the terminals can send messages at the same time, as long as they do not choose the same destination. However, the crossbar complexity is quadratic with respect to the number of terminals connected to it.

To balance both aspects, performance and technology limitations, a *multi-stage* network is often employed as the number of terminals increases. In these networks, the terminals are connected to switching points (usually called *routers*), which are in turn, connected to other routers within the network. These networks employ routers with a technologically feasible crossbar size. However, they do not provide a point-to-point connection between the terminals and then, the messages exchanged between them has to travel through one or more routers. An example of one possible implementation of a multi-stage network is presented in Figure 1-2.

As it can be inferred, each interconnection network offers different advantages and drawbacks, so computer engineers must work within technology constraints to meet the performance requirements of a system to implement the following four common aspects in the design of any interconnection network.

- *Topology*: it is the connection pattern between the switching points of the network, which describes precisely how these network nodes are interconnected by cables. Furthermore, the topology and how a computing system is packaged are closely related; typical packaging schemes are hierarchical. Topology is a critical feature of any interconnection network since that sets its performance bounds, by determining its *diameter* and *network bisection*, and its *path diversity*. The diameter of the network is

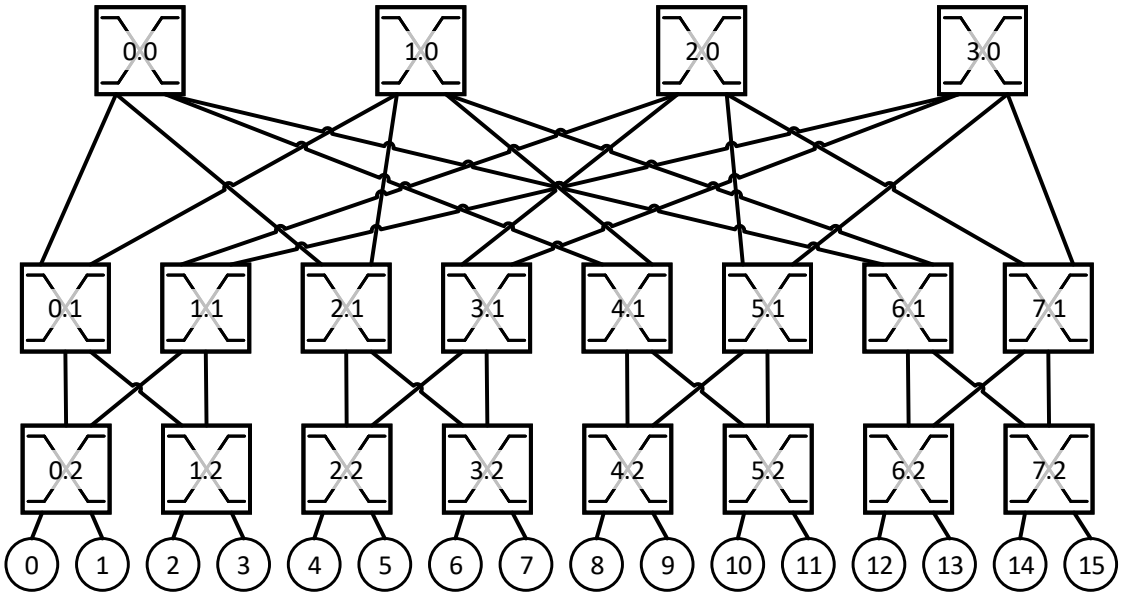


Figure 1-2: A multi-stage *folded-Clos* [47] interconnection network indexing the routers by their position in the *X* and *Y* axis as *X.Y*, starting from top left corner.

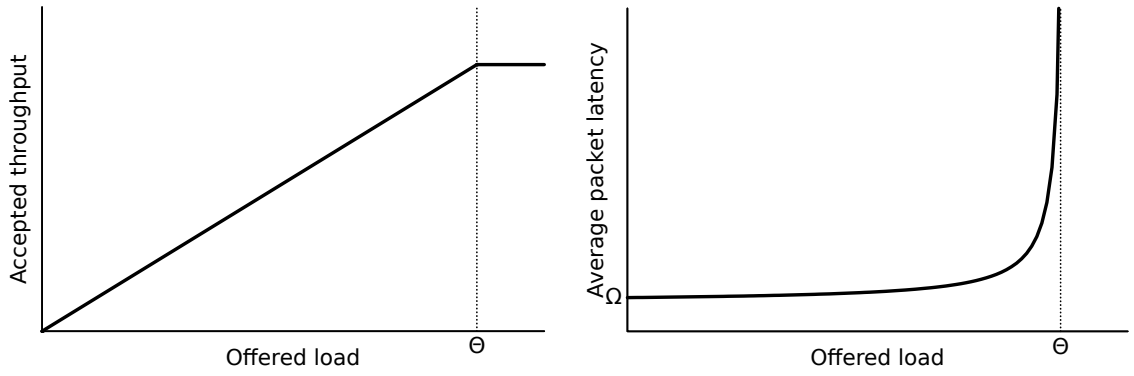
the largest hop count between any two routers when using minimal paths. Network bisection is the smallest set of channels that divide the network into two equal parts. Path diversity determines the number of possible minimal paths between most pairs of terminals. The network topology increases its impact over the performance of the applications that are distributed over the system as long as the link bandwidth increases and the router latency decreases.

- *Routing*: it determines which of these possible paths, either *minimal* or *non-minimal*, a message actually takes from the source to its destination. The routing algorithm determines how much of the performance bounds offered by the topology is achieved and balances load on the network under adversarial traffic patterns. The latter is mainly done by *adaptive* routing algorithms which select between minimal and non-minimal routes based on the network status.

- *Flow control*: it allocates the different resources of the network (mainly buffers tied to router ports) to packets while those are traversing the network. This aspect becomes more critical as the utilization of resources increases and can also prevent the *deadlock* and *livelock* pathologies in the network. Flow control mechanisms employ certain information to notify the upstream router that the receiver is not able to receive more messages. This information can indicate the availability like *pauses* or the amount of free space in the downstream router buffer like *credits*. Usually, these mechanisms are applied hop-by-hop but there are similar mechanisms to manage the congestion at end-to-end level like *explicit congestion notifications*.

- *Router architecture*: it defines the router organization, such as the switch fabric and buffers, and the control logic associated to these components. This is very tied to the routing and the flow control which will be implemented in the interconnection network. Routers may include certain indicators as *counters* available to both the network administrator and to themselves for being applied, for example to routing algorithms.

Each of the aforementioned aspects, which are analyzed in depth in Chapter 2, determines how the interconnection network is implemented and establishes a bound on its performance and its cost. The performance of an interconnection network is measured mainly by two quantities: the *throughput*, which indicates the sustained data transfer rate that is effectively achieved, and the *latency*, which measures the delay of messages between its source and its destination. In the typical performance graphs which represent throughput and latency, two zones can be delimited using the *saturation point* (Θ). At offered loads below Θ, the steady-state performance is characterized by a linear increase on both metrics and beyond that point, by a plain throughput[2] and an "infinite" latency as it is depicted in Figure 1-3. Throughput equals the offered load following a straight line below *saturation point*. This point Θ, is the highest value of offered load for which throughput, or accepted load, equals to it. The latency curve starts at the zero-load latency value (Ω) and experiences an exponential growth just before saturation.



(a) Throughput (b) Latency

Figure 1-3: Conceptual representation of (a) throughput and (b) latency performance metrics as a function of offered load.

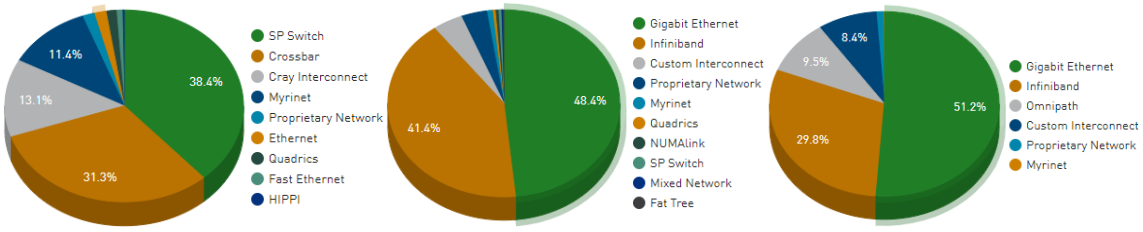
Technology evolution has led to a convergence in HPC and DC systems.[3] The former ones, driven by HPC applications, are often communication-intensive and can be sensitive to network latency. The latter ones, typically oriented to provide Internet Services such as web search or email, are softly insensitive to latency since they operate on human timescales.

[2] The throughput does not always continue plain for greater loads than Θ, as it is explained in Section 3.1.1.

[3] A good example of this convergence regarding interconnection networks is the Cray Slingshot [166] architecture. This recent proposal adds custom protocols to achieve high-performance and make Ethernet more suitable for HPC workloads while maintaining compatibility with standard Ethernet network devices, allowing it to be used efficiently in both HPC and DC environments.

However, the back-end applications providing those services may indeed require intensive communications and low latency characteristics. This fact, combined with the emergence of cloud computing and data-intensive applications insinuate that modern DC requirements look like HPC requirements. This thesis is mainly focused on HPC systems, however its contributions can be applied to large DC interconnection networks.

Traditionally, commodity *Ethernet* technology has been employed in domestic and business DC environments. However, Ethernet is taking up a significant portion in the HPC market with slightly more than half of the systems on the current Top500 list using any implementation of commodity Ethernet as shown in Figure 1-4(c). *InfiniBand* [153], in all of its variants, accounts for 30% of the interconnections leaving very little room for proprietary and custom networks. From its introduction in HPC systems in the early 2000s, Ethernet technology has been used in a significant fraction of the systems; in the 2000s, only about 2% of the Top500 systems employed Ethernet; in 2010, it was nearly 48% and nowadays it is about 53%. Moreover, in the current list, after InfiniBand, the third interconnection family with a 9.8% of the systems is *Omni-Path* [32],[4] which is not commodity but it is also Ethernet technology.



(a) 2000 (b) 2010 (c) 2020

Figure 1-4: Breakdown of supercomputer interconnect family from the last three decades of June Top500 lists. Ethernet technology is highlighted each represented year. Note that the colors employed in the figures do not represent the same technology in all of them.
Source: `https://www.top500.org/statistics/list/` **© Top500.org**

Regarding the programmability and following the trends motivated by the development of cloud computing, the concept of *Software-Defined Networking* (SDN) appeared. SDN is a network architecture approach which aims to make networks agile and flexible, enabling the network to be intelligently and centrally controlled ("programmed") using software applications. SDN suggests the use of a centralized controller that directly controls multiple forwarding components in the network. Furthermore, this concept continues evolving and its next step has been enabling the definition network devices' behavior through a configuration language independent of the target, while the control of operations is still centralized. These network devices are known as *protocol independent switch architecture* devices, and they claim to not incur on any power or cost penalty compared to fixed-functions network devices and to be a domain-specific processor for networking.

[4]Intel has recently discontinued it and it will not continue the development of Omni-Path networks.

1.2 Objectives

In light of the above, the first aspect to define an interconnection network for a computing system is the topology, which is tied to system packaging constraints. Typically, power and cost-efficient scalable networks with low diameter rely on topologies that approach the Moore bound[5] [82, 135] and, in order to get the maximum size, they lack minimal path diversity. Hence, once the topology is defined and, therefore, the performance bounds of the network are determined, a suitable routing algorithm should be designed to accomplish as much as possible of those limits and due to the lack of minimal path diversity it must exploit non-minimal paths when the traffic pattern is adversarial. These adaptive routing algorithms select between minimal and non-minimal paths depending on the network conditions, which are typically based on comparisons of the flow-control credits of different output ports. The non-minimal paths of such routings are usually built according to Valiant's proposal [187, 186] to load-balancing the network links, which implies that the non-minimal path doubles the length of minimal path, and then increases the latency supported by packets.

The aim of this PhD work, which reaches its highest point in this dissertation, is the design of non-minimal adaptive routing algorithms for efficient interconnection networks. This objective is approached by dividing it in two orthogonal sub-objectives. The fist is motivated by the Mont-Blanc project [157], which designed and built supercomputers based on ARM SoCs[6] for mobile environments. These SoCs usually have an Ethernet network interface and, due to the lack of flow-control credits in Ethernet, the goal is to design a routing capable of selecting between minimal or non-minimal paths without relying on credits. The second is motivated by the existence of multiple non-minimal paths between routers in low-diameter topologies [54], such as Flattened Butterfly [110] or Dragonfly [108], and the possibility of adapting the routing to network conditions [54, 110, 108]. Hence, the goal is to adapt the number of hops in the non-minimal paths to the traffic conditions instead of blindly using paths according to Valiant load-balancing algorithm.

1.3 Major dissertation contributions

The most remarkable contributions of this thesis to the study of non-minimal adaptive routings for efficient interconnection networks are listed next. Each of them is presented in depth in different chapters of this document.

- In *HPC networking over commodity Ethernet technology*, a realistic and competitive implementation of a scalable lossless commodity Ethernet network for exascale-level

[5] The Moore bound indicates the maximum number of nodes that a graph can have for a given node degree and diameter.

[6] *System on a Chip* (SoC).

HPC environments is introduced, considering low-diameter and low-power topologies. It is also proposed a hierarchical routing based on location-dependent MAC addresses, TCAM rules compaction and conditional OpenFlow rules, which exploit the explicit congestion notifications that travel through the network to decide between using minimal and non-minimal paths to forward each packet. This avoids the dependence of flow-control credits, which are not available in commodity Ethernet technology, to balance the network load. This contribution was sided with the idea of Mont-Blanc project [157], which implied the use of commodity Ethernet for HPC.

- In *non-minimal adaptive routing with latency improvements*, two mechanisms to improve the non-minimal path of an adaptive routing are proposed to reduce the average latency and increase throughput. It is also proposed a routing which leverages the aforementioned mechanisms to reduce path length for local and global traffic. This routing relies on a sequence of policies ordered by path length and on a hysteresis mechanism to improve the performance and avoid variability. This routing can be combined with other non-minimal routing mechanisms, such as Piggyback. In this case, the proposed mechanism decides the path length of the non-minimal route and the base mechanism selects between minimal or non-minimal for each packet.

- In *latency-optimized non-minimal adaptive routing for Dragonfly networks*, a topology-dependent optimized routing for Dragonfly networks is presented. The proposed routing algorithm determines a path for the packets at injection, which introduces the minimum amount of additional hops while avoids network congestion based on the network conditions and on the inferred traffic pattern. It extends the traffic counters already present in modern routers to adapt non-minimal path lengths to the traffic pattern, and modulates the UGAL selection of paths based on live network conditions to bias its decision to minimal or non-minimal route for each packet.

1.4 Outline

For the remainder of this dissertation, immediately following this introduction, Chapter 2 provides some background on interconnection networks. Chapter 3 presents the aspects of the employed methodology, common to the whole dissertation. Chapters 4, 5 and 6 explore the three main contributions of this work which have been presented on the previous section. The first one delves into an implementation of a commodity Ethernet network for HPC environments with a novel routing which leverages the explicit congestion notifications presented in the network to decide between minimal and non-minimal path for each packet. The other two chapters present different mechanisms, one agnostic and other dependent of

the topology, to adapt the number of hops that each packet must follow based on the network status. Chapter 7 presents the most notable related work to the proposals presented in this dissertation and analyzes its similarities and differences. Finally, some conclusions, future lines of work and the list of publications written during the PhD are presented in Chapter 8.

2 Background

This chapter aims to provide a better understanding about different and fundamental aspects of interconnection networks that are relevant to this PhD dissertation. Section 2.1 starts with the necessary details about router architecture, followed by an introduction, in Section 2.2, to system level topologies and particularly to the Dragonfly topology. The next relevant topic, explained in Section 2.3, is the routing. Then, Section 2.4 introduces the flow control and deadlock and livelock issues are introduced in Section 2.5. Finally, the software-defined networking concept is explained in Section 2.6.

Chapter contents

2.1 Router

Routers[1] as switching nodes are the building blocks of the interconnection networks. Three are the fundamental tasks of the routers: 1) to determine the best path for a packet through the network to its destination; 2) to actually forward packets received on its inputs through the appropriate output; and 3) to temporarily store received packets in its buffers to absorb traffic bursts and temporary congestion before forwarding them.

The size of a router, measured as its number of ports, is denoted as *router radix* (k) or *router degree*. Historically, large-scale interconnection networks were built based on low-radix routers [51, 5], however, advances in signaling technology and the exponential increase of the pin bandwidth have enabled bigger routers. Indeed, Kim *et al.* [107] indicate that it is more cost-effective to partition this increment of the pin bandwidth building high-radix routers with thin channels than building low-radix routers with fat channels. One of the first systems that has used a high-radix router, concretely with 64 ports in each YARC router [164], was the Cray BlackWidow [3].

Routers can be logically partitioned basically in three *planes* or layers: the *data* plane, which refers to all processes that forward packets from one interface to another; the *control* plane, which refers to all functions that determine how packets should be forwarded; and the *management* plane, which refers to all methods that can be used to configure the control plane and to monitor the network device. Each of the router planes, in turn, is divided into different elements. The data plane or also called *data-path* is composed by memory elements as buffers at the input and/or output ports and a switching fabric, on the other hand, the control plane basically consists of combinational logic circuits grouped as different functional units like *routing computation*, *virtual channel allocation* and *switch allocation*. The management plane is omitted because its functionalities are not related to this dissertation. A block diagram exhibiting the aforementioned parts of a router is shown in Figure 2-1. It is composed by a series of buffers at the input and output ports, a switching fabric and the functional units within the control plane.

The switching fabric is the key component of the router's architecture and is composed by an $N \times M$ *crossbar* and its respective scheduler. Commonly, symmetric $N \times N$ crossbars are employed. The crossbar implements in a non-blocking way all permutations of connections among its N input and N output ports at switching points, denoted as *crosspoints*. Each of them can be turned on or off by the scheduler taking into account that only one of the N inputs can be spatially connected to at most one of the N outputs at each time step. A basic crossbar is a 2×2 bufferless switch and takes this name because it could be in *cross* state, connecting input 1 to output 2 and input 2 to output 1, or in *bar* state, connecting input 1 to output 1 and input 2 to output 2 [168]. This basic concept has been extended to switches implementing any input to output permutation with multiple inputs and outputs. Designing

[1] Equally denoted as *switches* during this dissertation.

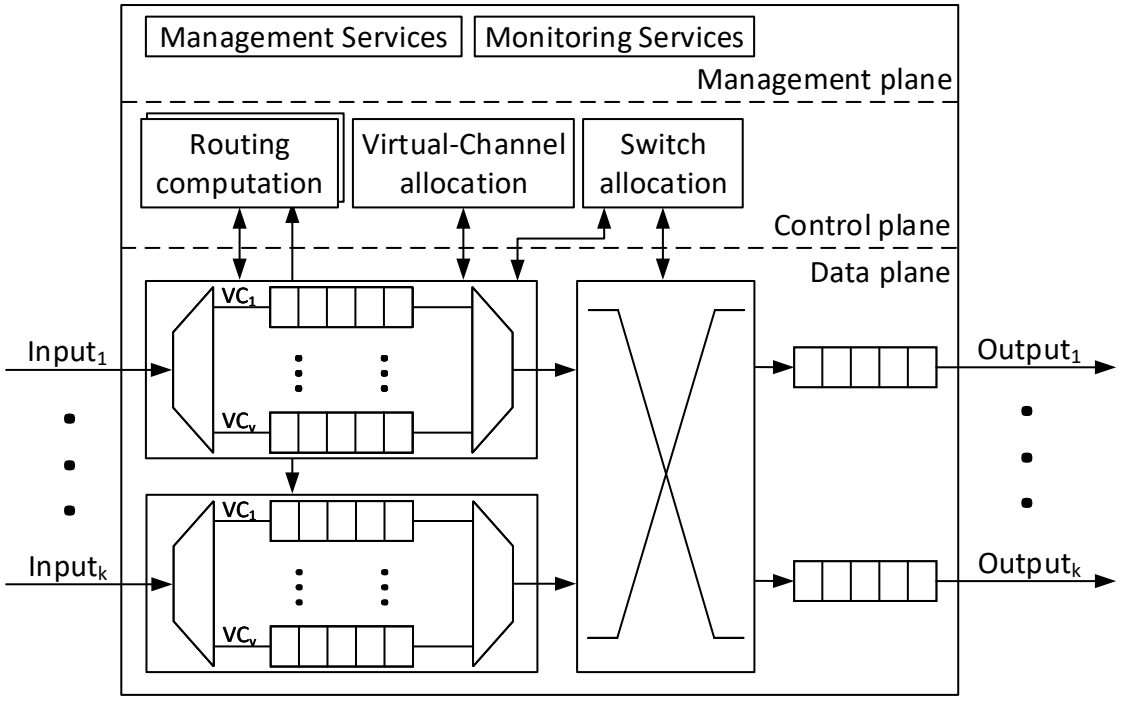


Figure 2-1: Planes and components of a virtual-channel router with input and output buffering.

a switching fabric for a router based on a bufferless monolithic crossbar is simple but expensive in terms of scheduling complexity, although it was an implementation option used formerly [128, 41, 191] and also recently [32, 57]. On the other hand, fully-buffered crossbar designs, which add a buffer to all crosspoints, or partially-buffered crossbar switching fabrics, which maintain a few separate buffers per output, have been considered an implementation option to improve the throughput over bufferless crossbars [140, 160, 2, 133]. A more sophisticated design, usually employed to build large switching matrices, is a multi-staged fabric compounded of multiple smaller crossbar switches [42, 164]. This work hereon takes a bufferless monolithic crossbar implementation option as the reference crossbar.

For bufferless crossbar switches, there is no buffer at the crosspoints. However, buffers may be placed at the input side, output side or both, categorized respectively as *Input-Queued* (IQ) switches [102], *Output-Queued* (OQ) switches [102], and *Combined Input-Output Queued* (CIOQ) switches [155]. Figure 2-2 portraits a scheme of each of the above-mentioned different queuing strategies. The buffers associated with each channel are typically *Firs-In, First-Out* (FIFO) queues where the oldest packet, or the *head* of the queue is the first leaving the buffer. Traditionally, the most frequent buffering strategy has been the output-queued and is attractive because it may achieve 100% throughput. However, since there are no queues at the inputs, the crossbar should take a packet from each input port in each clock cycle, being also possible that more than one packet should be delivered through the same output port. Indeed, for a crossbar with N inputs, there could be up to N packets destined to the same output port simultaneously. Hence, to be capable of receiving all

the packets at one time slot, the internal interconnection and memory need a bandwidth of N packets per time slot, where a time slot is defined as the time between packet arrivals at input ports. This requirement is denoted as the *internal speed-up* of the switch [43]. This concept is analyzed latter in this section. Due to the increment of the routers' radix and the required port bandwidth, this output buffering strategy is not practical because of their very high memory bandwidth requirement. On the other hand, the internal interconnection and memory bandwidth is not a problem with the input queuing strategy because they need only run as fast as the line rate. This is very appealing for high-radix routers or routers with high bandwidth ports [128, 149].

(a) IQ (b) OQ (c) CIOQ

Figure 2-2: Scheme of (a) input, (b) output and (c) combined input-output queuing strategies for a bufferless crossbar switching fabric.

However, this queuing strategy can suffer from *Head-of-Line* (HoL , [102, 182]) blocking and the result presented by Karol *et al.* [102], which limits the throughput due to HoL blocking to 58.6% if each input maintains a single FIFO, nearly invalidates input queuing strategy. This limitation was under uniform traffic but it is even worse under periodic traffic patterns [121]. To solve the HoL blocking problem, an increase in the internal speed-up of the router has been proposed. If the switching fabric has a speed-up between 1 (IQ) and N (OQ), for example 2, which doubles the line rate, packets need to be buffered at the inputs before switching as well as at the outputs after switching. This is known as CIOQ queuing strategy. Prabhakar *et al.* [155] postulate that a CIOQ switch with a speed-up of 4 can behave identically to a FIFO OQ switch. Note that this queuing strategy provides the best throughput. Moreover, Anderson *et al.* [14] show that the impact of HoL blocking problem in input queues can be completely eliminated by replacing the FIFO queuing discipline with *Virtual Output Queuing* (VOQ, [177]) architecture at the input side, hence a throughput of 100% can be achieved without speed-up under random uniform traffic pattern. VOQ architecture maintains at each input port a separate queue for each one of the N output ports as shown in Figure 2-3.

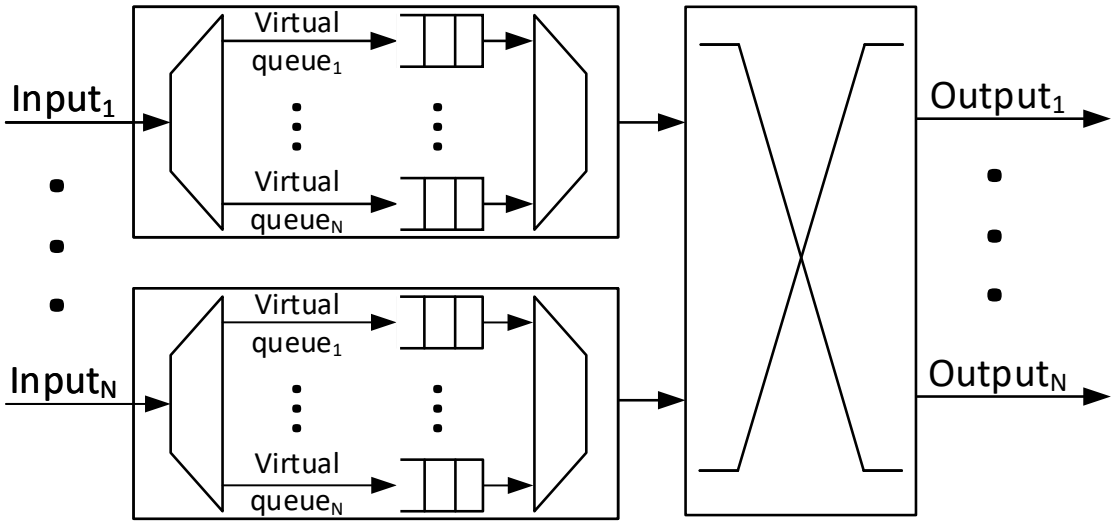


Figure 2-3: Virtual output queuing architecture.

The *Virtual Channel* (VC) concept, firstly introduced by Baubold *et al.* [17], consists in a logical division of the physical link into different partitions or virtual channels. Each of them has an independent buffer and flow control from the rest of VCs that share the physical link. The VCs allow a decoupling between the allocation of buffers and the channels by providing several buffers for each network link at the cost of additional control logic. This decoupling allows active packets to overtake blocked packets which can alleviate the HoL blocking problem and increase the network throughput. Furthermore, VCs can be used to prevent deadlock as it will be presented in Section 2.5 and to provide multiple levels of quality of service by separating different classes of traffic into different VCs. As it can be seen in Figure 2-1, which portraits the reference router architecture employed in this work, the buffers of virtual channels are connected to the same input network link through a demultiplexer and to switch by a multiplexer.

Another very constrictive aspect of the router architecture is the type of *flow control* employed. Flow control is a mechanism used to determine how the resources of a network are allocated to the packets which are traversing it. Its main goal is to use resources efficiently to achieve the best possible network performance. Flow control can be divided into bufferless or buffered [54]. Section 2.4 explores in depth flow-control mechanisms and their possible implementations.

The primary task of the router is the routing itself, which is the process of determining the path for a packet trough the network from its source to its destination. This task is done by the *Routing Computation* (RC) logical unit within the control plane of the router. Section 2.3 goes in depth into the different routing mechanisms. However, the relevant aspect of routing concerning router architecture is the implementation of the routing computation, typically known as *routing mechanics* [54]. This can be implemented using either *algorithmic* or a *table-based* structure. The algorithmic way employs a fixed logic circuits for deter-

mining the output port based on the current router and the destination information. The outcome of this implementation option can be very simple but also a very inflexible routing algorithm. Indeed, this option is usually restricted to simple routing algorithms and regular topologies. On the other hand, a very adaptable implementation is the table-based one, which implements a lookup table for obtaining the appropriate output port for the packets based on the destination information. This table is supported by a *Content-Addressable Memory* (CAM, [113]) or by a Ternary-CAM (TCAM, [188]) when longest-prefix matching is required, such as in IP routing.

CAM memories allow simultaneous comparison between all indexes and the key. The main advantage of this type of memories is that the search time is bounded by one single memory access, hence, a high lookup throughput is guaranteed. Binary CAMs support storage and searching of data where each bit has two possible states: zero or one (0 or 1), whereas ternary CAMs allow each bit of data to be either a zero, a one, or a *wildcard* (0, 1 or *X*). Hence, binary CAMs perform exact-match bit-by-bit searches while the *X* input, which is often referred to as a "don't care" state, enables TCAMs to perform broader searches based on pattern matching. Then, binary CAMs allow only fixed-length comparisons and are not directly capable of doing varying-length lookups. To cope with the longest-prefix matching requirement, TCAMs can be used directly [125] or in a hybrid scheme combining binary and ternary CAMs [176]. TCAM memories store a separate mask for each index, and during a lookup, both the key and the indexes are masked before its comparison. Both CAM and TCAM are commercially available and can be used in a discrete form or embedded with the rest of the system in a single device. However, both of them are much more costly than conventional memory like RAM, and they also consume a great deal of power, so they dissipate a lot of heat.

The *Virtual channel Allocation* (VA) and *Switch Allocation* (SA) logical units perform a matching between a group of router resources and a group of requesters, basically matching input buffers to the output ports requested so that one flit from each input port and one flit destined to each output port, at most, are selected. There are multiple allocation algorithms although this work only considers *separable allocators* [54, 19, 18], which perform allocation by decomposing allocation into two successive stages of arbitration, one across the inputs and another across the outputs. This arbitration can be performed in either order, in an *input-first* or *output-first* separable allocator. In the former, an arbitration first selects a single request at each input port and later, the outputs of these input arbiters are input to a set of output arbiters to select a single request for each output port. In the latter, the output arbitration is performed first and then the input arbitration. The result of separable allocators ensures at most one grant asserted for each input and for each output. However, an optimal matching is not guaranteed because it can leave an input and an output which could have been trivially connected, both idle. Nevertheless, this type of allocators can be implemented easily and perform a fast matching, specially on high-radix routers. Remembering the Fig-

ure 2-1, note that in each input port there is one buffer per every virtual channel. The VA unit or input arbitration selects one of the VCs for each input port of the crossbar and places a request for the corresponding output port, and then, SA unit or output arbitration matches each output port to an input that has requested it. Each arbiter employs an arbitration policy to set the order in which the requests should be attended. These policies should ensure a fair attendance to avoid starvation at certain inputs. One of the most used policies is *Round Robin* (RR). The router modeled in this work employs an input-first separable allocator using RR in each arbiter, updating the priority list of ports only when that arbiter generates a winning grant as it is done by iSLIP [127].

Most routers split their functionality into multiple basic pipelined steps, which could be performed concurrently for different packets, to optimize the clock cycle and increase the performance of the network device. Following the model introduced by Dally *et al.* [54] for a typical virtual-channel router, the pipeline of the router consists on four steps. These can be separated into *per-packet* and *per-flit* steps, where a flit is a unit on which the packets are broken, which will be analyzed more in depth in Section 2.4. The former ones are the *routing computation* at which output port is selected based on the information stored in the packet's header; and the *virtual channel allocation* at which the packet arbitration is performed. The latter ones are the *switch allocation*, at which output arbitration matches each output port to an input that has requested it; and the *Switch Traversal* (ST) at which flits that have won the access to the crossbar are transferred from their input buffers to output buffers, where flits will be waiting up until the corresponding channel is ready for going through it. Figure 2-4 portraits the pipeline of these four steps for two packets during nine cycles of the router.

Data \ Cycle		1	2	3	4	5	6	7	8	9
Packet 1	Head flit	RC	VA	SA	ST					
	Body flit				SA	ST				
	Tail flit					SA	ST			
Packet 2	Head flit				RC	VA	SA	ST		
	Body flit							SA	ST	
	Tail flit								SA	ST

Figure 2-4: Two packets traversing the pipeline of the modeled router.

With the goals of increasing the router performance and improving the sub-optimal performance of resource allocation, the crossbar can have a certain *speed-up*, especially with small flit sizes and low latency requirements [54]. The crossbar speed-up can be provided at the input, at the output or be an internal speed-up. Input speed-up option is provided in space, that is, the crossbar has s several input ports for each of k router's input port, resulting in a crossbar with $s \cdot k$ input ports. In the same way, output speed-up imply that the crossbar

has s several output ports for each of k router's output ports. On the other hand, internal speed-up is provided in time, that is, the crossbar has higher bandwidth or frequency than router's input and output ports. This increment in the frequency of the crossbar also impacts on the logic control units. The router modeled in this work employs an internal speed-up where the four pipelined stages work at a higher frequency than the network links.

2.2 System level topology

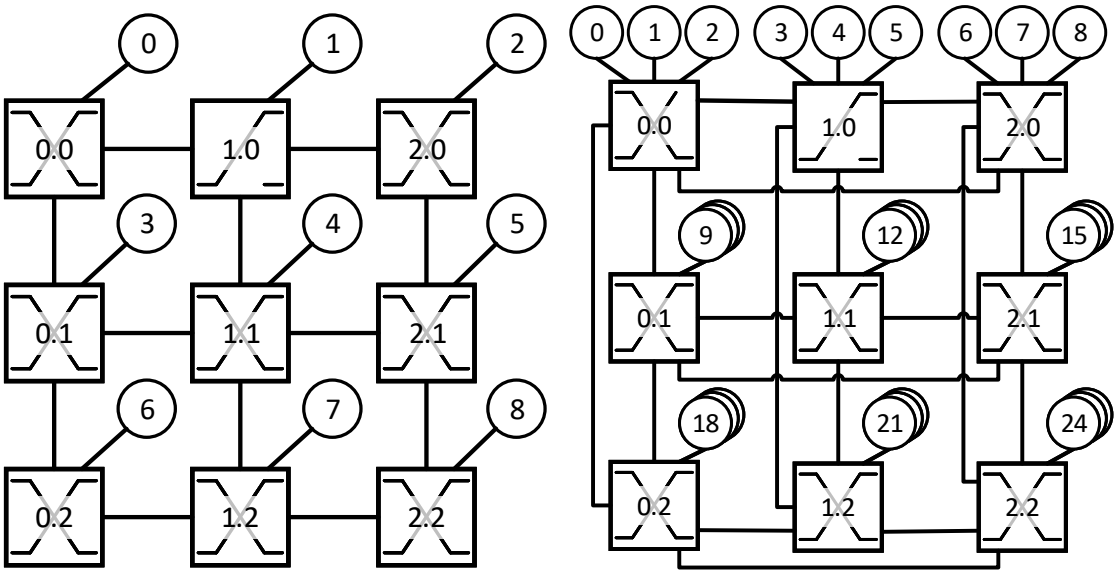
An interconnection network is composed by a set of routers connected to other routers and to computing hosts by network links in a particular pattern. This pattern is known as the network *topology*, which describes precisely how the switching points of the network are interconnected and determines the *path diversity*. The path diversity is the number of disjoint paths or routes between each source and destination pair of terminals within the network. This feature allows the topologies to tolerate faulty channels without the isolation of computing hosts and a better load balance across network links. No topology is optimal since its design involves different trade-offs, such in terms of performance, cost, wiring complexity, and resilience. As introduced previously, an interconnection network can be used mainly at two levels; *system-level* and *on-chip*. Several topologies can be used in both domains, however, this work is focused in system level networks and so, when a topology is employed, always is from a point of view of an interconnection network which connects computing nodes with another ones.

The selected topology for an interconnection network mostly determines its performance and cost [54] and its choice is mainly driven by three factors: the *required performance*, *technology* and *system packaging*. Ideally, the topology selected for a system should ensure the performance requirements of the applications with the minimum possible cost. Bounds of the main performance metrics, *throughput* and *latency*, can be deduced analyzing the network topology by graph theory. To do that, the routers are identified with the vertices of a graph and its edges are identified with the communication links of the network. The technology of a topology refers to the basis of the router assembly (i.e., power, pin density, radix, etc) and the signaling technology, such as electrical or optical links. The system packaging decides the distribution of the computation resources per unit of area in the *Data Center* (DC) and it also determines the signaling speed due to the distance over which the communication is reliable. A good match that fit in the system packing into the network topology simplifies the system wiring and gives a cost-effective solution.

Topologies can be broken down into two large groups: *direct* and *indirect* [171, 54]. In direct topologies there are no *transit* routers that are switching nodes without any attached compute host.[2] Traditionally, in a direct topology there was the same number of computing nodes as routers, and each of the former is connected to one of the latter. In this previ-

[2]In contrast, a router with terminals directly attached to it is denoted as *access*.

ous scenario, commonly, the router and the terminal formed a block and the packets were forwarded directly between these blocks. Nowadays, the direct topology concept allows a *concentration* factor which states the number of terminals that are connected to a single router. Hence, an updated definition for direct topology includes topologies on which every router within the network is connected at least to one or more terminals as well as to other switches. This work will hereon uses this updated definition with regard to direct topologies. On the other hand, in an indirect topology there are transit routers, then, some switching nodes are connected only to the neighboring switches. Figure 2-5 portraits an example of two direct topologies where the switch nodes are shown as squares and the terminals as circles. An example of an indirect topology can be found in Figure 1-2. Select between a direct or an indirect topology determines some packaging and cabling requirements as well as fault resilience and impacts on the network cost.



(a) Mesh (b) 2-D Flattened Butterfly

Figure 2-5: Conceptual representation of (a) mesh and (b) bidimensional Flattened Butterfly direct interconnection networks. The concentration factors are 1 and 3 respectively. The routers are indexed by their position in the *X* and *Y* axis as *X.Y*, starting from top left corner.

An important characteristic of a topology is its *diameter*, which is the maximum distance, measured as the number of communication links that a packet must traverse between any source and destination routers when using minimal paths.[3] This characteristic of the topology bounds the maximum or *worst-case zero-load*[4] latency of the network. Note that the network diameter is independent of traffic pattern and is dependent solely of the topology.

[3]The network diameter can also be measured as the number of router traversals along the longest minimal path within the network. This work does not employ this definition for the diameter characteristic.

[4]The zero-load or *base* latency represents the average latency suffered by a packet in absence of other network traffic. Hence, the packet does not contend for network resources with other packets.

The most popular topologies for DCs are folded-Clos networks [47], which are also known as *Fat-trees* [120]. They are hierarchical multi-level and indirect networks and are known to perform well for all kinds of traffic patterns. However, they are relatively costly due to the amount of routers and network links that they require and it becomes prohibitively expensive for large deployments. Torus, which is also a popular topology for HPC systems, directly interconnect a terminal to a number of its neighbors in a k-dimensional lattice [54]. Its main advantage is its cost and it adapts very well to problems that requires a neighbor communication pattern. However, Tori topologies provide low network throughput for adversary traffic patterns and its implementation is not easy with some system packaging constraints and cable length limitations due to signaling speed. Besta *et al.* [28] introduce the SlimFly topology to provide a trade-off between the cost and the throughput. It is a direct topology that approximate the Moore bound, in fact, it reaches approximately the 88% of it for a diameter of 2. HyperX topology [9] is a regular, multi-dimensional topology and it is a generalization of the hypercube topology [30]. To balance the network and achieve full global bandwidth, the number of routers on each dimension must be equal and equal to the number of terminals connected to each router. However, this is not a requisite and some configurations may be not balanced. A HyperX with the same number of routers in each dimension is also known as Flattened Butterfly topology (FB, [110]), which is a Hamming graph [139]. Dragonfly direct topology [108] is a hierarchical network with two levels: *intra-group* and *inter-group*. An arbitrary topology may be selected for each level. It is described in depth in the further section. The Megafly topology [64] (also known as Dragonfly+ [170]) is a Dragonfly in which the intra-group topology is a two-level Fat-tree while the inter-group is a completely-connected topology. The topology in each group is also known as a bipartite connected graph which is a graph with two subgroups where all nodes within each subgroup are connected to all nodes in the another but there is no connection within a subgroup. This combination of topologies allows higher scalability than Dragonfly based on completely-connected topology on both levels while reduces the cost compared to a Fat-tree.

Regarding the parameter used to indicate the router's size, network topologies can be constructed with low-radix or high-radix routers. Technology trends motivated the use of low-radix topologies, such as the *k*-ary *n*-cubes [51, 5], in the 1990s and 2000s by several *High-Performance Computing* (HPC) systems such as the 3-D torus of the Cray T3E [165], the SGI Origin 2000 hypercube [118], the 2-D torus of the Alpha 21364 [138], the 3-D torus of the Cray XT3 [36] and the IBM Blue Gene/P [85]. In low-radix topologies, the routers are often only connected to neighboring routers which benefit to applications with near-neighbor communication patterns. However, the technology improvements, such as the increasing pin bandwidth or the advent of economical optical signaling [108], have motivated the migration towards high-radix topologies, such as the *folded-Clos*, *Dragonfly*, *Dragonfly+* and *HyperX* or *Flattened Butterfly* networks. For example, the Cray BlackWidow [3], Summit [184] and Sierra [119] systems employ a Fat-tree network [175], the Dragonfly and

Dragonfly+ topologies are used in Trinity [122], which is an implementation of Cray XC architecture [60] using Aries interconnection network [13] and Niagara [154] systems respectively, and Domke *et al.* [58] have deployed the first machine based on HyperX topology.

High-radix topologies exploit the technology evolution by reducing the network diameter and hence, obtaining a lower latency and cost than previous proposals. Moreover, some recently proposed topologies exploit the evolution of the technology to improve cost-effectiveness. For example, Flattened Butterfly topology, reduces the cost of the network by approximately 50% from a folded-Clos topology by removing intermediate stages (routers and channels) and creating a direct network [110]. However, it is ultimately limited by the physical constraints of a router radix; and the cost of scaling to large node count increases the number of dimensions, which implies an increment in costs and latency. To cope with these limitations, a set of routers can be used together to create a *virtual high-radix router*. The *Dragonfly* topology [108], analyzed in depth below, leverages this concept to create a more scalable and power efficient topology.

2.2.1 Dragonfly

Kim *et al.* [108] first introduced the Dragonfly topology focused on scalability and cost-efficiency. It is designed from the premise that intra-group topology can be interconnected using cheap electrical wires, while optical cables are used for longer inter-group connections. The Dragonfly extends the savings of FB and it reduces the overall cost of the network by approximately 10% over FB for networks up to 4K terminals and by approximately 20% for larger networks [108]. It is a direct low-diameter topology deployed hierarchically in two levels which adapt to a suitable system packaging. Figure 2-6 shows a block diagram of a network with 72 compute hosts using this topology.

The first level comprises sets (or *groups*) of routers interconnected following certain *intra-group* topology by *local* communication links, each of them can be treated as a virtual high-radix router. These groups are connected by *global* communication links according to an *inter-group* topology. Arbitrary networks can be used in both topological levels. For the intra-group topology, Cray has implemented in the Cascade (XC) systems [60, 13] a rectangular 2-D HyperX topology in which the routers within a group are arranged in logical rows and columns and connected in an all-to-all fashion between the routers belonging to each row or column. On the other hand, IBM in the PERCS system [15] and recently Cray in its Shasta exascale supercomputer system platform using the Slingshot network [166] have implemented the intra-group topology using a complete graph,[5] in which each router among a group is connected to every other router by a direct network link. For the inter-group topology, all aforementioned systems employ a complete graph (direct all-to-all) interconnection topology between groups. Additionally, the connection between each pair of groups can be

[5] Slingshot network design allows to arrange the switches arbitrarily but its default topology is a Dragonfly with an all-to-all interconnection pattern in both topological levels [166].

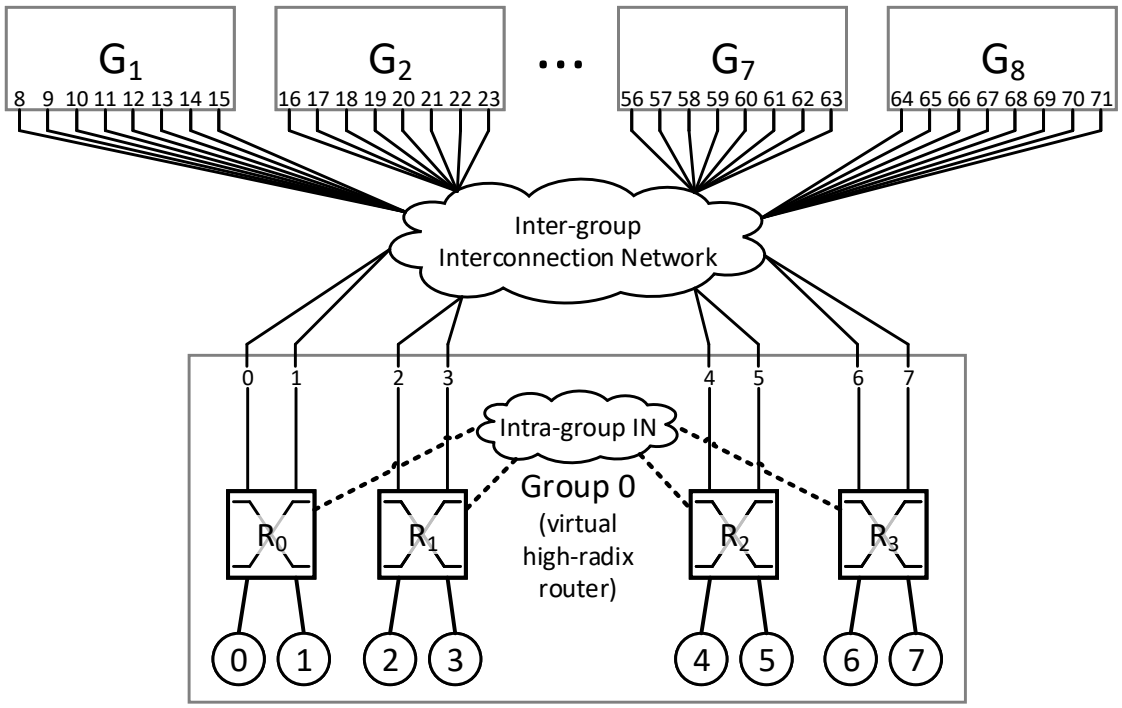


Figure 2-6: Block diagram of a Dragonfly topology (*h=2, p=2, a=4*) remarking the concept of virtual high-radix router, introduced by this topology, and its two hierarchical levels.

done by one or multiple global links, latter possibility is denoted as *global trunking*. A Dragonfly network using complete graphs in both topological levels is denoted by Camarero *et al.* [38] as a *canonical* Dragonfly. Like in the base version of the Dragonfly and in the most recent systems which use this topology, a completely-connected graph is used in this work for interconnecting routers within each group and between groups. Hence, hereon any reference to a Dragonfly network will regard to a canonical Dragonfly unless otherwise noted in a particular context.

This topology can be described through three parameters: the number of compute hosts connected to each router p, the number of routers per group a, and the quantity of global or inter-group links per router h. Hence, the number of ports per router or router radix must be, at least, equal to $k = p+a-1+h$. Each group has $a \cdot p$ links to compute hosts and $a \cdot h$ global links to other groups, acting as a virtual router with radix $k' = a \cdot (p + h)$. The Dragonfly network parameters p, a and h can have any value. However, to ensure a balanced use of the communication links under a load-balanced traffic, these parameters must follow the relation $a = 2p = 2h$. This equals the number of global links departing from a group to the number of compute hosts or injectors, and presents the same proportion between global and local links as the hops required by minimal paths. Even if maintaining a balanced Dragonfly topology is not a necessary requirement, this work always assumes it because any level of imbalance results in a wasted capacity on some links.

With a single link between pairs of groups and so, without global trunking, it is generated a Dragonfly network with $G = 1 + a \cdot h = 2h^2 + 1$ groups achieving its maximum-size $N_{MAX} = a \cdot p \cdot G = 2h \cdot (2h^2 + 1) = 2h^3 + 2h$ terminals. For example, an $h = 6$ Dragonfly has $p = 6$ compute hosts per router, $a = 12$ routers per group and $G = 73$ groups, for an overall $N = N_{MAX} = 5,256$ compute hosts. This maximum size implies that there is no minimal path diversity, that is, only one minimal path connects any pair of given compute hosts. And based on that, it is simple to observe that the diameter of a Dragonfly network is three (one hop in each group plus one hop in the global interconnect), corresponding to the longest minimal path between two switching nodes, which occurs when these routers belong to different groups and none of them is directly connected to the global link between the groups.

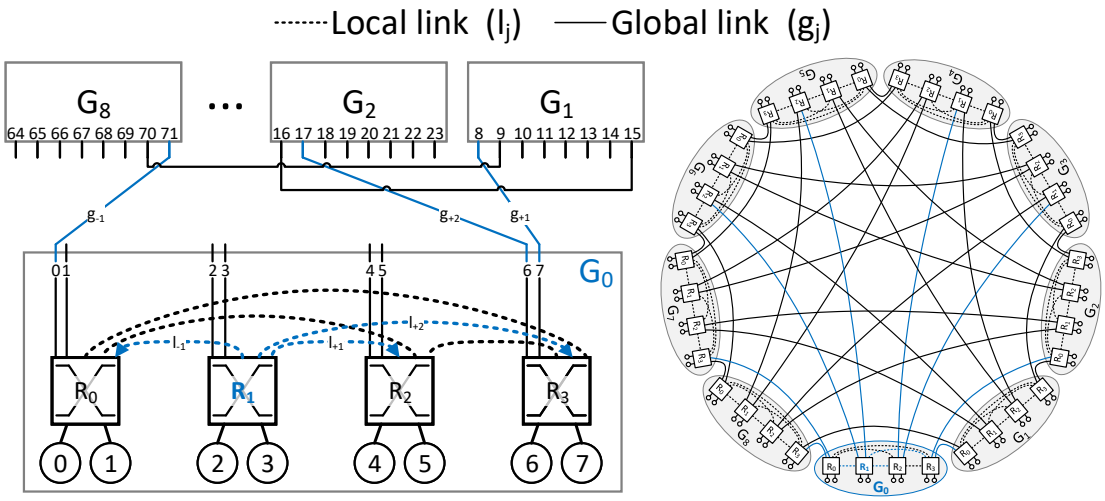
Table 2-1 summarizes the symbols used in this work regarding the Dragonfly topology. The groups are denoted as $G_0, G_1, \ldots, G_{2h^2}$ and are typically depicted in a circle following a counterclockwise order. The routers in the groups are denoted as $R_0, R_1, \ldots, R_{a-1}$ and are typically depicted within the group from left to right. The local link *leaving* from router R_i to router R_{i+j} modulo a is denoted as l_j ($j \in \{-h, -(h-1), \ldots, -1\} \cup \{+1, \ldots, +(h-1), +h\}$). Note that l_{+h} and l_{-h} both denote the same local link, and that local link l_{+i} is denoted as l_{-i} in the opposite direction.

Table 2-1: List of symbols employed in the topological description of Dragonfly.

Symbol	Description
p	Number of compute nodes per router
a	Number of routers per group
h	Number of inter-group (global) network links per router
k	Router radix (its number of ports)
k'	Virtual router radix
G	Number of groups in the network
N	Number of compute hosts in the network
N_{MAX}	Maximum number of terminals in the network

Kim *et al.* [108] compare the cost of electrical wires and active optical cables as a function of distance, and conclude that beyond ten meters the optical signaling is more economical. The Dragonfly topology exploits this relation between cost and distance employing electrical wires within a group and optical ones for links between groups. These global links can be arranged following multiple approaches [38, 80, 20] and research on Dragonfly systems often does not specify which arrangement is employed, in fact that property was not explicitly in-

dicated when Dragonfly was first proposed. However, the *global link arrangement* implicitly shown in the figures of that work is denoted as *consecutive* [38] or *absolute* [80, 20] arrangement. It connects the egress global links of a group, in consecutive order, to each other group in a sequential manner starting from group zero, except when the link would connect a group to itself. This arrangement results in a connection through the fist global link egressing each source group and group zero. A similar arrangement, which can be inferred from the figures in [72] and denoted a posteriori as *palmtree* [38] or *relative* [80, 20], connects each router R_j in group G_i to the h routers R_{a-1-j} in groups $i+h\cdot j+1, i+h\cdot j+2, \cdots, i+h\cdot j+h$ modulo G using h consecutive global links per router. Hence, the first router of a group is connected to the h precedent groups and the latest one to the h following groups. This last arrangement is like the previous one if the routers position within the group are reversed and with each group taking the role of group zero. Hence, each group presents the same global connectivity pattern.[6] Figure 2-7 depicts a block diagram, including the notation for local communication links, and a complete diagram of a canonical Dragonfly network with ($N = N_{MAX} = 72$) employing the palmtree global link arrangement. This work hereon takes palmtree as the reference global link arrangement.



(a) Block diagram denoting local and global links.

(b) Complete network.

Figure 2-7: Representation of a sample canonical Dragonfly network (*h=2, p=2, a=4*) depicting (a) a block diagram and (b) the complete network using the palmtree global link arrangement. Note that local and global links egressing from R_1 and G_0, respectively, are highlighted in blue.

Figure 2-8 presents the largest system that can be constructed using k-radix routers for different balanced network topologies. The number next to each topology name represents its network diameter. The Dragonfly topology, in any of its variants, can support the highest number of compute hosts compare to other topologies with the same diameter. For example, based on routers with $k = 64$ ports, a Flattened Butterfly with three dimensions scales to 65,536 compute hosts whereas a canonical Dragonfly reach up to 262,656 terminals with

[6] This rotational symmetry is specially leveraged by this work in Chapter 6.

a network diameter of three hops. A three-level folded-Clos and four-dimension Flattened Butterfly reach up to 65,536 and 371,293 compute hosts, respectively. The latest case is more scalable than a canonical Dragonfly at the cost of increasing the diameter. A 5-D Flattened Butterfly scales up to 1,771,561 terminals. However, a comparable Dragonfly with a diameter of five, like the proposed in the Cray Cascade systems, reaches up to 16,781,312 compute hosts.

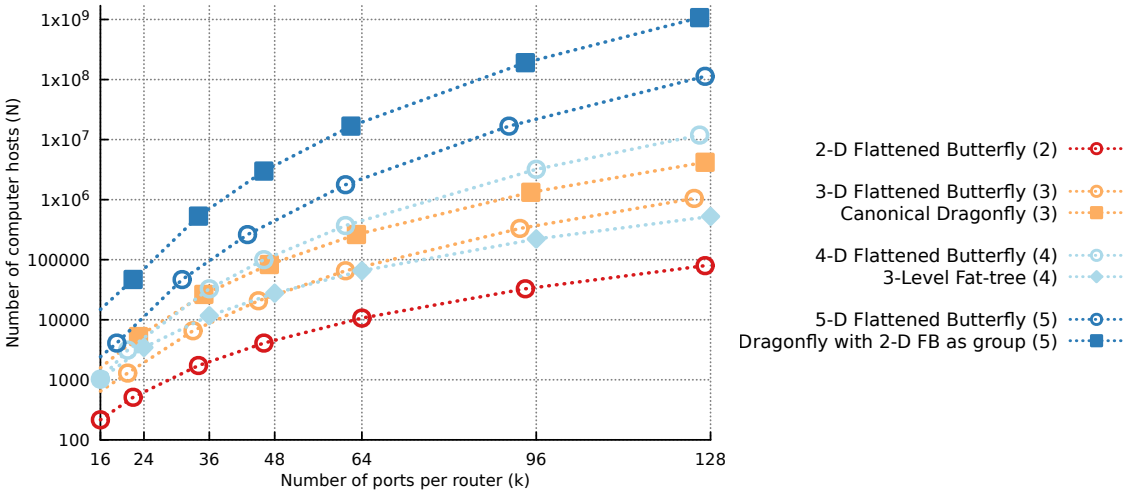


Figure 2-8: Scalability for different network topologies, measured as the number of compute hosts, as router radix *k* increases. Note that the number next to each topology name represents its network diameter and the ordinate axis uses a logarithmic scale.

The constant network link speed evolution has reduced the reach of copper cables and increased its price. Then, if the physical constraints of router radix are overcome and the cost of active optical cables is reduced further to be slightly higher than the cost of electrical wires or new link-level technologies allow the use of low-cost passive optical cables, a more flattened topology such as Flattened Butterfly can result in a more cost-efficient topology than a hierarchical one such as the Dragonfly. McDonald *et al.* [126] reproduce the model of costs introduced in [108] to compare it with confidential quotes gathered from multiple vendors for passive optical cables and conclude that the HyperX cost is always lower or equal to a Dragonfly topology.

The current trend towards massive-scale HPC data centers has driven the introduction of low-diameter, highly scalable network topologies such as Dragonfly. This kind of networks sacrifice minimal-length path diversity, common in traditional topologies such as folded-Clos, for larger number of nodes and lower average distance. Consequently, minimal paths are congestion-prone under adversarial traffic patterns and the use of non-minimal routes, which are typically controlled by an adaptive non-minimal routing algorithm, is mandatory in order to fully exploit the benefits of this topology compared to other high-radix topologies such as Fat-tree. The next section delves into the background on this topic.

2.3 Routing

Once routers and the communication links are ready, the next logical step is to determine how a packet should travel through them. Then, *routing* determines the path followed by a packet through the network from its source to its destination. As mentioned in Section 2.2, the topology determines the performance bounds of a network, even if routing is one of the key factors which determines how much of this ideal performance is achieved [54]. Furthermore, routing is responsible for exploiting all the features of topologies, such as the path diversity, and desirably balancing load across network links even in the presence of adversarial traffic patterns.

The RC unit of a router gives an output port for packets executing a specific *routing algorithm*, which should pursue different objectives sometimes confronted. A routing algorithm should exploit the path diversity of the topology, which can include both minimal and non-minimal paths, while keeping the path length as short as possible. The latter is pursued to use the minimal amount of network resources for delivering each packet and balancing the load across network links to achieve a high throughput under both, benign and adversarial traffic patterns. Moreover it should do all of this being complexity-effective to minimize its impact on packet latency, that is, it must be able to be implemented efficiently.

Routing algorithms can be classified according to different features [54, 78].

- Based on the hop count, they can be divided into *minimal* and *non-minimal*. The first ones select paths with a minimal number of links to be traversed between source and destination. Depending on the topology and its minimal path diversity, there can be only one or multiple minimal paths. The latter ones select paths between the source and destination of a packet that traverse more channels than the minimal paths between those source and destination points.

- According to its adaptivity, they can be classified into *oblivious*[7] and *adaptive* depending, respectively, on whether they ignore or use the state of the network for making routing decisions. The latter ones can be further divided into *congestion-oblivious* and *congestion-aware* based on whether they take or not the output link demand into account during the routing computation. Congestion-aware can be further divided into *local* and *regional/global* based on the source of the congestion information employed, which can be purely local or collected from other points in the network. Moreover, these other points can belong to a *region*, e.g., a group of Dragonfly topology, or be *global*, which implies that the information is shared to all the network by the source of the information.

- Based on the stage where the routing decision is taken, they can be divided into *source*

[7] Traditional *deterministic* routing algorithms, which always give the same network path for each pair of source and destination compute hosts, can also be classified as oblivious [54].

and *in-transit* (also denoted as *per-hop*, *incremental* or *progressive*) routing. The first ones determine the path at the source router[8] so, this computation is performed once per packet. The latter ones re-evaluate the routing algorithm at every hop through the followed path to determine the next productive hop for each packet to reach its destination. Hence, the routing computation is done multiple times for each packet by in-transit routing algorithms. This incremental adaptability allows a better sense of congestion because it can be encountered during the traversal of the network and it might not be observed at the source router during the injection of packets. This type of routing must set certain restrictions in the amount of adaptivity to ensure the forward progress of packets. However, the flexibility obtained by this type of routings is at the cost of a higher routing algorithm complexity. This work does not focus on this type of routing algorithms because it aims to maintain the routing complexity limited.

The following sections explore in depth a set of routing algorithms representative of each aforementioned categories, except in-transit, explaining its details for the Dragonfly topology.

2.3.1 Minimal (MIN)

The *Minimal* routing algorithm (MIN) selects the shortest path for a packet between its source and destination. It is minimum, so it keeps the path length as short as possible. In a Dragonfly network there is no minimal path diversity, so it can be implemented only as oblivious. And regarding the last classification option, it can be computed at source or hop-by-hop, although it is usually implemented at the source router.

In a general case, a MIN computation in a Dragonfly network from a source computing host S connected to a router R_S belonging to group G_S to a destination terminal D connected to a switching node R_D belonging to group G_D results in a single one possible path which consists of up to three hops through network links: a *local* hop in G_S, a mandatory *global* hop to the G_D, and a final *local* hop to the R_D. This route is usually represented in the longest way as lgl, or $l_1g_1l_2$[9] for referring to each hop individually. Shorter paths may occur, depending on the relative location of the source and destination routers. Let's split two different situations, intra-group and inter-group routing. In the first one, only the first local hop can be employed and solely if R_S and R_D are different. In inter-group routing, the global hop (g) is mandatory and the first local hop (l_1) is employed when R_S is non-adjacent to G_D, so that local hop is employed to reach another router R_P in G_S which is connected through a global link to G_D. The last local hop (l_2) is employed when the global link taken is not connected to R_D and

[8] Please do not confuse with another terminology where *source routing* implies that the source computing node of the packet prepends the complete route on each packet header. [33]

[9] Note that these subscripts do not have sign (+, −) rather than the notation used to indicate the Dragonfly *local* network links.

in this case, this global link reaches to router R_C in G_D and that local hop is employed to traverse from R_C to R_D. An example of different MIN routing computations are presented in Figure 2-9.

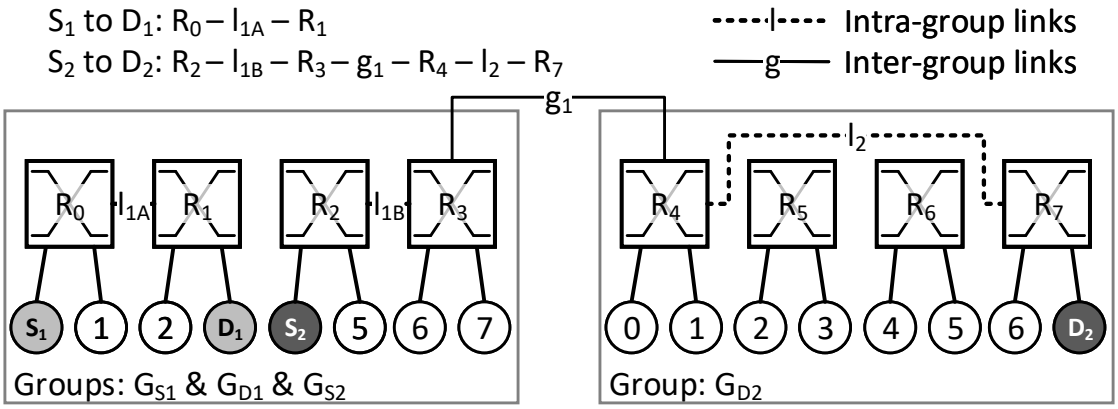


Figure 2-9: Two MIN routing computation examples. For example, in path from S_2 to D_2 the R_P and R_C are the routers R_3 and R_4, respectively.

MIN provides maximum throughput and minimum latency under uniformly distributed traffic loads, but under adversarial loads it saturates some network links resulting in a very poor performance. To alleviate this congestion concern, a non-minimal routing is desirable.

2.3.2 Valiant Load-Balancing (VLB)

Valiant load-balancing routing algorithm (VLB), also denoted as *Valiant randomized routing* algorithm and typically known as *Valiant routing*, is an *oblivious non-minimal* routing algorithm computed only at source, which was proposed by Valiant *et al.* [187]. VLB uses randomization and non-minimal routing to exploit the path diversity of the topology and achieve a load-balanced workload over the network. The original randomization-based Valiant algorithm [187, 186]) obtains $O(\log N)$ packet delivery time for any permutation of traffic in a hypercube network of N processors. It is a two-phase algorithm with an *intermediate* router (R_{ROOT}, [60, 64]) selected randomly among the routers in the network during routing computation. In the first phase (denoted as *phase A*), minimal routing is used to send each packet to the intermediate router. Once the packet reaches the intermediate switch, in the second phase (denoted as *phase B*) the packet is sent minimally from R_{ROOT} to the actual destination. The original implementation of phase A in [187] follows a dimension-order process where the (only) link in each dimension of the hypercube in each current router is traversed or not with the same random probability. This is equivalent to selecting a random intermediate destination at injection time, but requires less bookkeeping since the intermediate node is not recorded in the packet header. Valiant routing is completely *oblivious*: the path for each packet is selected randomly and it does not depend on the status of network nor the destinations of packets sent by other nodes. If the intermedi-

ate router happens to be either R_S or R_D, VLB degenerates into minimal routing as only one phase is performed.

VLB routing can be used in the Dragonfly topology to load-balance adversarial traffic patterns. Using the above-mentioned symbols, a VLB computation in a Dragonfly network obtains a route with up to six hops ($l_1g_1l_2 - l_3g_2l_4$), three on each phase. Note that there are two R_P and R_C routers in each complete VLB path because there is one of them in each VLB phase of MIN routing. Again, as explained for the MIN routing, shorter paths are possible depending on the relative location of the source, destination and intermediate routers. Ignoring the phase B, the VLB non-minimal path is determined by two aspects: 1) the global links egressing from G_S that can be used, restricting the groups that can be selected as the G_{ROOT}[10]; 2) the routers belonging to G_{ROOT} that can be selected as the intermediate switch R_{ROOT}. These two aspects can be combined to generate different *VLB phase A policies* that determine the routers of the network which can be selected as intermediate router R_{ROOT}.

Kim *et al.* [108] suggested that applying Valiant's algorithm selecting the router in the intermediate group that is adjacent to the source group, this implies that $R_{ROOT} = R_C$ following the notation introduced, suffices to balance load on both the global and local links. Hence, VLB was first implemented by those authors in the Dragonfly selecting a random intermediate group, instead of a router, while saving one local hop in the G_{ROOT}. Then, this VLB variant obtains paths with up to five hops ($l_1g_1 - l_3g_2l_4$). This implementation of VLB in the Dragonfly allows a performance increase under ADV+1 variant of the adversarial traffic which is presented in Section 3.2.2.1.2. However, shortening the path in phase A from its reference $l_1g_1l_2$ introduces pathological bottlenecks under certain adversarial traffic patterns. Garcia *et al.* [72] identified that the lack of the second local hop in VLB phase A introduces pathological performance issues under the ADV+*h* variant of the adversarial traffic. Similarly, as was identified by Fuentes *et al.* [68], non-minimal paths which omit the first local hop in phase A suffer under *adversarial consecutive* traffic pattern which is presented in Section 3.2.2.1.4. This work will hereon include the maximum hops presented in the phase A of the path in the abbreviation of VLB in order to propose diverse VLB path length options. Note that missing hops will be represented by a hyphen. An example of VLB_{lgl}, VLB_{lg-} and VLB_{-g-} routing computations are presented in Figure 2-10.

The randomization used by VLB effectively balances the load on the network links under non-uniform traffic loads. In fact, VLB provides optimal performance on adversarial traffic patterns regarding the throughput. This performance metric is limited on VLB to the 50% of the maximum throughput allowed by the network [181] caused by the double utilization of the network links. Moreover, due to the increment in the length of the path, VLB increases the zero-load latency and it also loses the traffic locality because a packet with an intra-group traffic pattern may be sent to a remote group by VLB.[11] VLB improves the performance un-

[10] Note that this restriction also restricts the possible routers that can be selected as R_{ROOT}.
[11] This aspect is partially mitigated in Chapter 5.

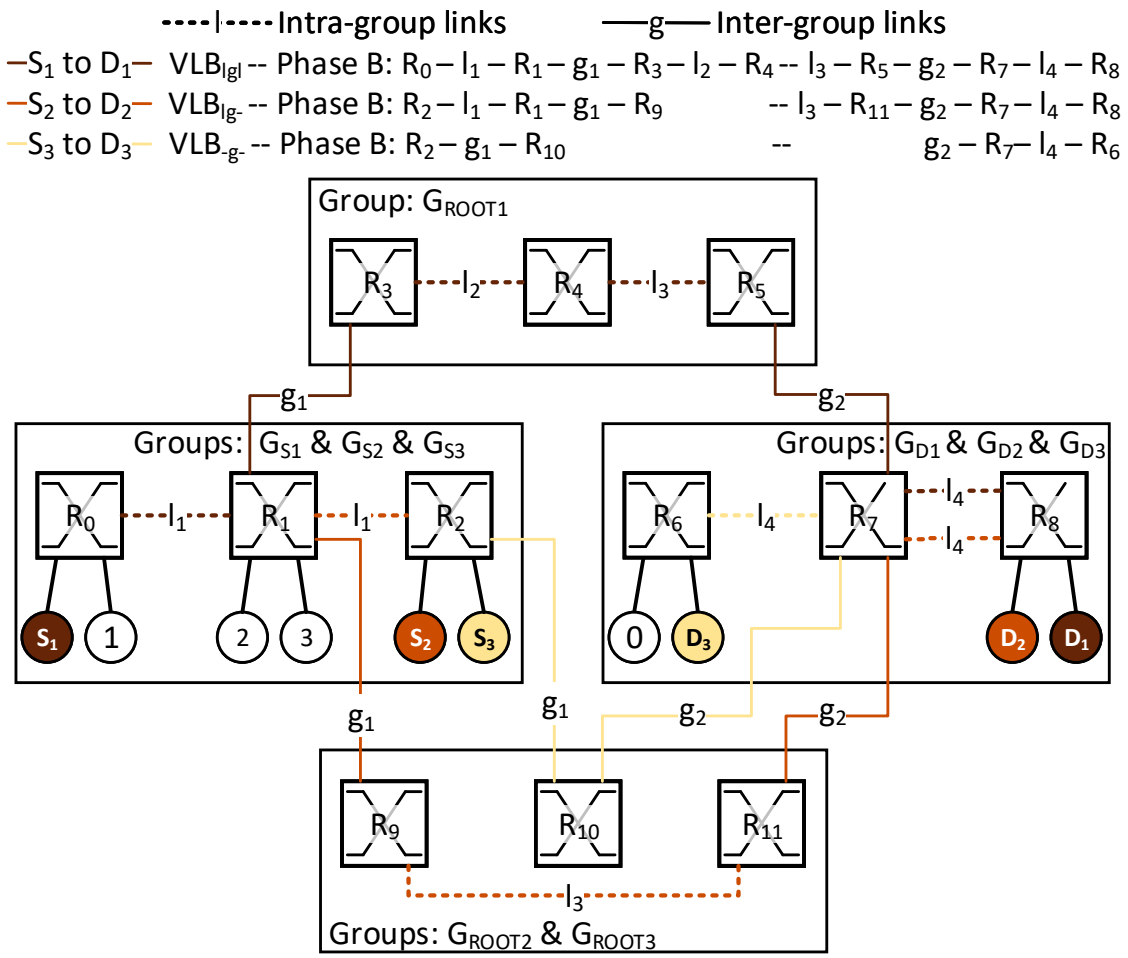


Figure 2-10: Three different VLB routing computations examples remarking source, intermediate and destination routers and the local and global links traversed. Note that S_3 to D_3 path is shorter than usual because the minimal path on phase B omits the l_3 hop. For example, in path from S_1 to D_1 the R_P and R_C in the VLB phase A path are the routers R_1 and R_3, respectively; and the R_{ROOT} is the R_4. The R_P and R_C in the VLB phase B path are the routers R_5 and R_7, respectively.

der non-uniform traffic loads, although under uniform loads it obtains half the throughput of MIN routing. To overcome this limitation, the following sections present some adaptive routing algorithms that dynamically decide between minimal and non-minimal routing to maximize performance.

2.3.3 Universal Globally Adaptive Load-balancing (UGAL)

Universal Globally Adaptive Load-balancing routing algorithm (UGAL) was proposed by Singh [172] to select between minimal and non-minimal routing for every packet based on the next hop congestion information (queue lengths) in each path. If non-minimal routing is chosen for a packet, this is routed as VLB, otherwise MIN routing is used. Under benign traffic patterns, UGAL aims at the performance of minimal routing whereas under adversarial traffic patterns, UGAL sends most of the traffic using VLB to load-balance the use of the

network links. Hence, UGAL is a *non-minimal source-adaptive congestion-aware*[12] routing algorithm.

To select between minimal and non-minimal routing UGAL uses the product of the queue occupancy (Q) and the hop count (H) for both routing paths. The minimal queue occupancy (Q_{MIN}) represents, in phits, the congestion on the output port taken by the minimal path while minimal hop count (H_{MIN}) is the number of hops between the source and destination of the packet following such minimal path. In the same way of VLB, UGAL selects a random intermediate router for the non-minimal path. Non-minimal queue occupancy (Q_{VLB}) represents the congestion on the output port taken by the minimal path between the source and the intermediate router while non-minimal hop count (H_{VLB}) is the sum of the hop count of the just mentioned minimal path and the hop count between the intermediate router and the actual destination of the packet. Based on that congestion information, UGAL routes a packet minimally if the following inequality is satisfied:

$$Q_{MIN} \times H_{MIN} \leq Q_{VLB} \times H_{VLB}.$$

Otherwise, the packet is routed non-minimally. Jiang *et al.* [96] implemented UGAL in the Dragonfly with minor changes from its original proposal. Since the global network links limit the network bandwidth and dominate network latency, the decision can be simplified using only the number of global hops (1 or 2 for MIN and VLB, respectively) instead of the hop count of minimal and non-minimal paths. Additionally, these authors added a routing threshold constant T which can be adjusted to bias UGAL towards MIN or VLB to balance performance between benign and adversarial traffic patterns and to filter out transient load imbalances on the minimal paths. Then, a packet is routed minimally if the following inequality is satisfied:

$$Q_{MIN} \leq 2 \times Q_{VLB} + T.$$

This work recovers the original comparison aforementioned to highlight the importance of local network links and preserves the threshold constant introduced in [96]. Hence, the implementation of UGAL employed by this work decides to route a packet minimally if the following inequality is fulfilled:

$$Q_{MIN} \times H_{MIN} \leq Q_{VLB} \times H_{VLB} + T. \tag{2-1}$$

The source router is usually not connected to the global network link to be used by MIN or VLB routing, therefore it must use the length of its local queues as a sign of their occupancy. This approximation is enough in networks with stiff back-pressure, but results in high latency in networks with soft back-pressure [96]. To cope with those situations, in which the

[12] Kim *et al.* [108] propose two implementations of UGAL for Dragonfly networks: one *local* congestion-aware and another *regional*, which are denoted as UGAL-L and UGAL-G respectively.

local congestion information may be insufficient to take a good routing decision, the congestion information of the channels can be exchanged between adjacent routers. This technique is employed by the routing algorithm presented in the next section.

2.3.4 Piggyback (PB)

Piggyback routing algorithm (PB) was proposed by Jiang *et al.* [96] as a complementary method to decide whether to route a packet minimally or non-minimally. It extends UGAL routing with a mechanism to detect saturated global links. PB routing inherits part of the classifications of UGAL, so it is a *source-adaptive* and *congestion-aware* routing algorithm. However, PB is classified as *regional* congestion-aware whereas UGAL is local congestion-aware, where a region is defined as a group.

PB broadcasts state information of the global network links to all adjacent routers. To do that, a link state bit vector is piggybacked[13] and broadcast periodically on idle local channels. The most recent version of this information for every global link of each group is maintained by each of their routers. Hence, PB decides to route a packet minimally only if the minimal global channel is uncongested and inequality presented in Equation 2-1 is fulfilled. Otherwise, the packet is routed non-minimally. The congestion state information of a global channel (CG) is represented into a single bit (C_{gc}) and calculated using

$$C_{gc} = Q_{gc} > 2 \times \bar{Q} + T,$$

where Q_{gc} is the queue occupancy of the global link, $\bar{Q}$ is the average queue occupancy of the other global channels of the router and T is a routing threshold to filter out transient load imbalances on the minimal paths. If C_{gc} is true, global network link gc is congested and should not be used to send a packet, otherwise it is considered uncongested. The original implementation of PB in [96] uses a fixed 2× factor but this work generalizes it to be able to use other values, resulting in the following employed equation:

$$C_{gc} = Q_{gc} > F \times \bar{Q} + T. \tag{2-2}$$

This routing algorithm is taken as the per-packet basis adaptive routing reference in this work. Note that both PB and UGAL are focused on avoiding in-network congestion and this work is not either focused on end-point congestion avoidance.

[13] Piggybacking, originally, is a technique to provide a better utilization of bandwidth sending the acknowledgments in outgoing data packets. Here, it implies that the state information is attached to the header of every packet exchanged within the group.

2.4 Flow control

Flow control is the basic mechanism by which the resources of a network, such as link bandwidth and buffer capacity, are managed and allocated to the packets and determines the solution when packets contend for such resources. Its main goal is to use resources efficiently to achieve the best possible network performance. The flow-control mechanism must avoid resource conflicts, because they are in demand by multiple uncoordinated sources, and allocates those network resources efficiently to achieve a high fraction of the ideal performance bounded by the topology and later by the routing algorithm. Flow control can be divided in *hop-by-hop* or *end-to-end*. As already mentioned, this work is focused on analyzing the performance of the network, so end-point congestion is not the focus. Hence, the end-to-end flow control which aims to mitigate this type of congestion, mainly by injection throttling, is not inside the scope of this work.

With respect to what can be done when a resource is requested but not currently available for use, the flow-control mechanisms can be divided into bufferless or buffered [54]. In the former implementation option the blocked packets can be either dropped or misrouted, and in the latter those packets are stored in buffers waiting for the availability of their demanded resource. An intermediate implementation option which require a bit more complexity is *circuit switching*, where only packet headers are buffered. This mechanism sends the header of a packet reserving the required resources through the network between its source and destination and once the path is completely reserved, the payload of the packet may be sent over the established circuit.

Employing buffers decouples the allocation of adjacent channels in time. A buffer is a place where the data that comes from one channel while waiting for the following one is stored, which allows the allocation of that second channel to be delayed. By contrast, without a buffer, the allocation of the two channels must be done during consecutive cycles, or the packet must be dropped or misrouted through another channel. Hence, the existence of buffers improves the flow control efficiency significantly. There are two main approaches for buffered flow-control mechanisms based on the granularity at which they perform resource allocation; it can be done in units of packets or at the finer granularity of flits. The breaking of large and usually variable-length packets into smaller fixed-size flits contributes to build the architecture of routers efficiently and to reduce the amount of storage needed in those network devices. There are two main flow-control techniques: *store-and-forward* [98] and *Virtual Cut-Through* (VCT, [105]). In store-and-forward, each switching node along a path in the network waits until a packet is completely received and stored in its buffer and then the packet is forwarded to the next node, whereas in VCT, the transmission of a packet to the next channel starts directly when the new header flit is received. *Wormhole* (WH, [53, 50, 52]) flow control operates like VCT, but managing the resources in units of flits rather than packets. WH reduces the amount of minimal buffering required on each router and due

to this smaller buffer size requirement it is commonly employed in memory-constrained architectures such as *networks-on-chip*, even though VCT is also employed in this type of networks [150]. The reduction of buffer sizes is also leveraged to build less power demanding and inexpensive devices. Due to the orientation of this thesis to system networks, where the memory is not a big problem, this work hereon takes virtual cut-through as the reference flow-control technique.

Buffered flow-control mechanisms must allocate the buffers, and to do that, need to track the buffer's status of the neighboring routers, either the buffer occupancy or the availability. This control information should be sent by flow-control mechanisms and is usually denoted as *backpressure*. Upstream routers are informed when they must stop transmitting because the downstream buffers are full. Different strategies exist to provide such backpressure [54], the options relevant for this work are examined in the following Paragraphs.

The *Xon/Xoff* or *stop/go* flow control [76] uses a single control status bit on the upstream router which represents whether the upstream router (or sender) is allowed to send (*on/go*) or not (*off/stop*) to a downstream router (or receiver). A signal is sent to sender only when it is necessary to change the control status. This mechanism uses two buffer occupancy thresholds (T_{off} and T_{on}) [189] to ensure that no data are dropped due to a lack of space at the receiver buffers and to be able to receive in-flight data sent during the propagation delay of the flow-control signal. The basic idea behind these thresholds is represented in Figure 2-11 and Figure 2-12 portraits the operation of this mechanism between three routers exchanging the Xon and Xoff signals.

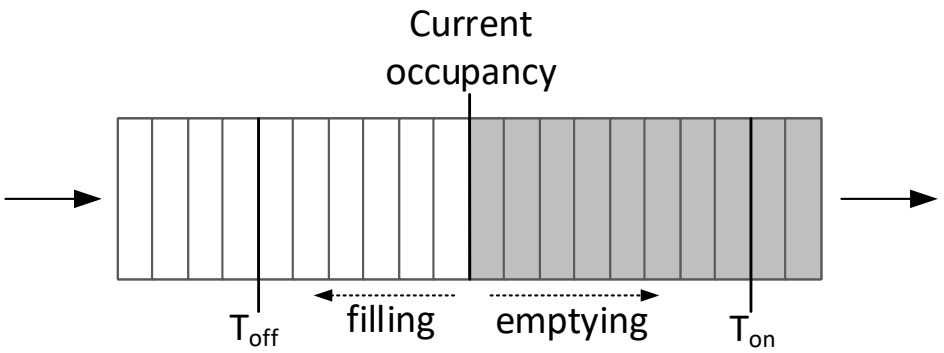


Figure 2-11: Basic idea behind the stop and go flow-control thresholds: T_{on} and T_{off}.

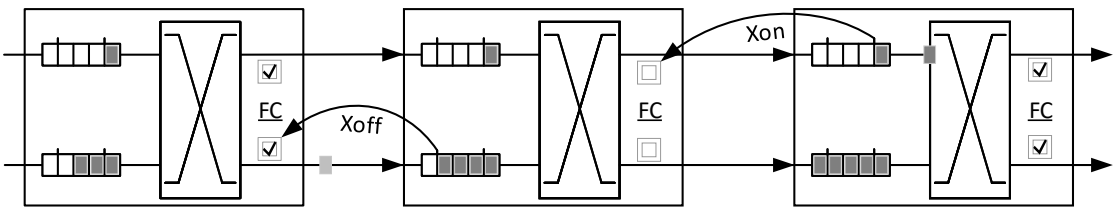


Figure 2-12: Example of Xoff and Xon signals sent by receiver to upstream routers.

The *credit*-based flow-control mechanism [116, 147, 167] is more efficient but less robust compared to previous mechanism. It is more efficient because requires the amount of space equivalent to one round trip time on buffers to achieve losslessness while stop/go requires the equivalent to two. On the other hand, Xon/Xoff flow control is a stateless protocol while the typical implementation of credits, which uses *relative* credit updates that inform about the availability to store additional data units, is a stateful protocol. The *credits* of an output on sender represent the amount of free space in the adjacent receiver's input buffer. In this flow-control mechanism, the upstream router keeps a counter of available credits updated when it sends a packet through the output or when the adjacent router removes a packet from their input buffer. That is, when the downstream router forwards a packet and frees space on its corresponding input buffer, it sends a credit to the upstream router, causing an increment in the upstream's credits counter. Figure 2-13 represent an example of these two mentioned actions. When a new packet arrives at the output port of the downstream router, it checks the credits counter and only sends the packet if there are enough credits on the output. This quantity is different for VCT and WH: whereas the latter requires only one credit, the first needs at least as many as the size of the packet. If previous condition is fulfilled for the corresponding flow control employed, the packet is sent and the credits counter is decremented. Once the counter reaches zero, downstream buffer is full and transmissions through this output are halted until the buffer becomes available.

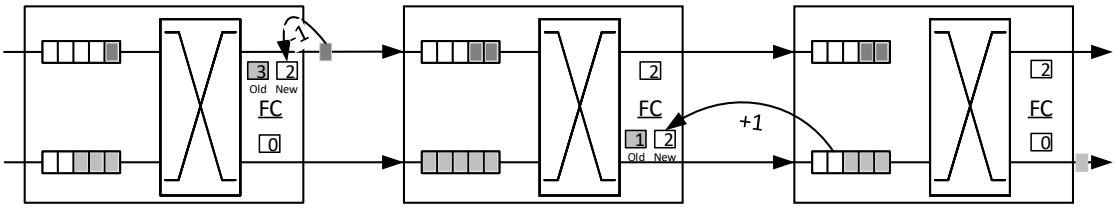


Figure 2-13: Credit-based flow control example remarking the relative updates of the credit counters after sending a packet by router and after signaling from the downstream router.

The backpressure signals can be transmitted on separate wires or, in a network with bidirectional links, they can be transmitted out-of-band on these channels in the opposite direction of the data by which have been triggered. If multiple VCs are used, like in the reference architecture of this work in Figure 2-1, the signaling must contains the VC index about it is notifying, so the backpressure traffic is incremented a bit.

2.4.1 Quantized Congestion Notification (QCN)

In systems, in which there is no communication between the congested devices and the source, detection of congestion in lossless networks relies on either buffer occupancy values or increases of flow latency. Alternatively, *Explicit Congestion Notifications* (ECN) can be generated by the network switches, typically triggered by the evolution of its buffers occu-

pancy. Such explicit notifications can be piggybacked on the data frames of the flow that suffers congestion, such as IP's ECN bits [158] of InfiniBand's FECN/BECN bits [152], or they can be independent messages such as the *Congestion Notification Messages* (CNMs) from QCN, which is described next.

Quantized Congestion Notification (QCN, [89]) implements congestion notification in Layer-2 Ethernet DC Networks. It is mainly composed of two elements, *Congestion Points* (CP, in switches) and *Reaction Points* (RP, rate limiters at NICs). CP samples the arriving frames according to a sampling interval and when a congestion situation is identified in a given buffer, it generates and sends an explicit *congestion notification message* to the source of the most recent frame (or flow), which is considered the culprit flow. This standard QCN's behavior is denoted by Neeser *et al.* [142] as *arrival sampling*. In contrast with arrival sampling, they propose the *occupancy sampling* method for CPs in which the CNM is sent to the source of a frame *randomly* picked from the whole buffer. When a CIOQ switch architecture is considered, occupancy sampling can be performed according to the two alternatives considered in Chapter 4 and described next.

- *Congestion detection at switch output buffers*: like in the original implementation of QCN [89], the sampling is done in the output buffers of the switch. However, instead of marking the most recent frame as the victim, this is randomly selected from the buffer. One advantage of sampling at output buffers is that the generated CNMs can be sent to all sources in the network regardless of their source switch.

- *Congestion detection at switch input buffers*: Neeser *et al.* [142] also suggest the use of input-buffer sampling as a congestion indicator to provide better fairness and improve the detection of offending and victim flows.

To identify a congested situation, two state variables are combined; position (Q_{off}) and velocity (Q_{δ}) of the sampled buffer occupancy, both variables are illustrated in Figure 2-14. CNMs do not indicate the mere presence of congestion, but instead they carry a feedback value (Fb) that quantifies it. A reference length denoted as *equilibrium point* Q_{eq} is assigned to each buffer. Q denotes the instantaneous buffer occupancy sampled every hundred packets and Q_{old} denotes the buffer length when the previous CNM was generated. Considering $Q_{off} = Q - Q_{eq}$, $Q_{\delta} = Q - Q_{old}$, and w a constant weight, set to two by default, Fb is calculated according to the following expression:

$$Fb = -(Q_{off} + w \times Q_{\delta}). \tag{2-3}$$

Negative values indicate the presence of congestion and generate CNMs. The Fb value in the CNM is quantized to 6 bits, ranging from 0 to 63, saturating with an occupancy of $2 \times Q_{eq}$. This value is sent to the source of the packet sampled in the switch buffer, this is, a source NIC. RPs at NICs implement injection throttling, using a mechanism based on an *additive-*

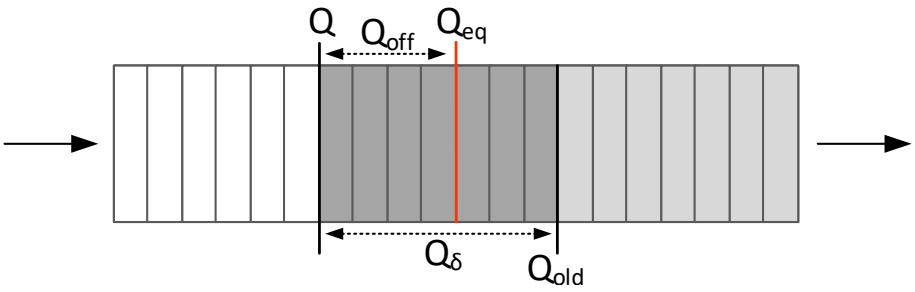


Figure 2-14: Representation of QCN buffer state variables at the congestion point.

increase, multiplicative-decrease policy. When negative congestion notifications are received, the feedback value *Fb* is used to divide the injection rate, adjusted by a factor L_f. QCN does not implement positive notifications (lack of congestion), so the additive increase policy is applied when no notifications are received for a period corresponding to a hundred frames.

2.5 Deadlock and livelock

Since there can be multiple packets in the network requesting different resources, which may be in use, a sequence of these packets can form a cycle. This situation can happen in almost all topologies, and they are prone to inter-lock these packets. This situation is usually known as *deadlock* [180] and are caused by a set of packets that can not advance because they are waiting on resources that are reserved by other packets in a circular chain, that is, there is a cycle of resource dependencies. Figure 2-15 shows an example of deadlock caused by a cyclic dependency on which packets are unable to make a left-hand turn. Deadlock is catastrophic for the network because another packets will be blocked on the same resources acquired by deadlocked packets and these, in turn, will block other packets until the whole network is flooded. Deadlock can appear in the network itself and due to dependencies in higher-level protocols [54], when there is a cyclic dependency between different packet classes that reuse the same network links. Once a deadlock appears on a network, it can only be broken by dropping a packet from the network. Lossless networks do not drop packets, so to prevent the network halt, networks must either use *deadlock avoidance* or *deadlock recovery* mechanism [54]. Deadlock avoidance mechanism guarantees that a deadlock situation can not occur, whereas deadlock recovery mechanism detects and recovers the network from a deadlock situation. Recovery mechanisms are not considered in this work, so the next paragraph presents the deadlock avoidance mechanism used.

Deadlock can be avoided by preventing the occurrence of cycles in the resource dependence graph generated by packets on the network. This can be done in a general way by numbering the resources and allocating them in an ascending (or descending) order. This general rule can imply routing or flow-control restrictions. A technique to break the cycles, which was considered unfeasible in the past due to the large amount of resources needed for

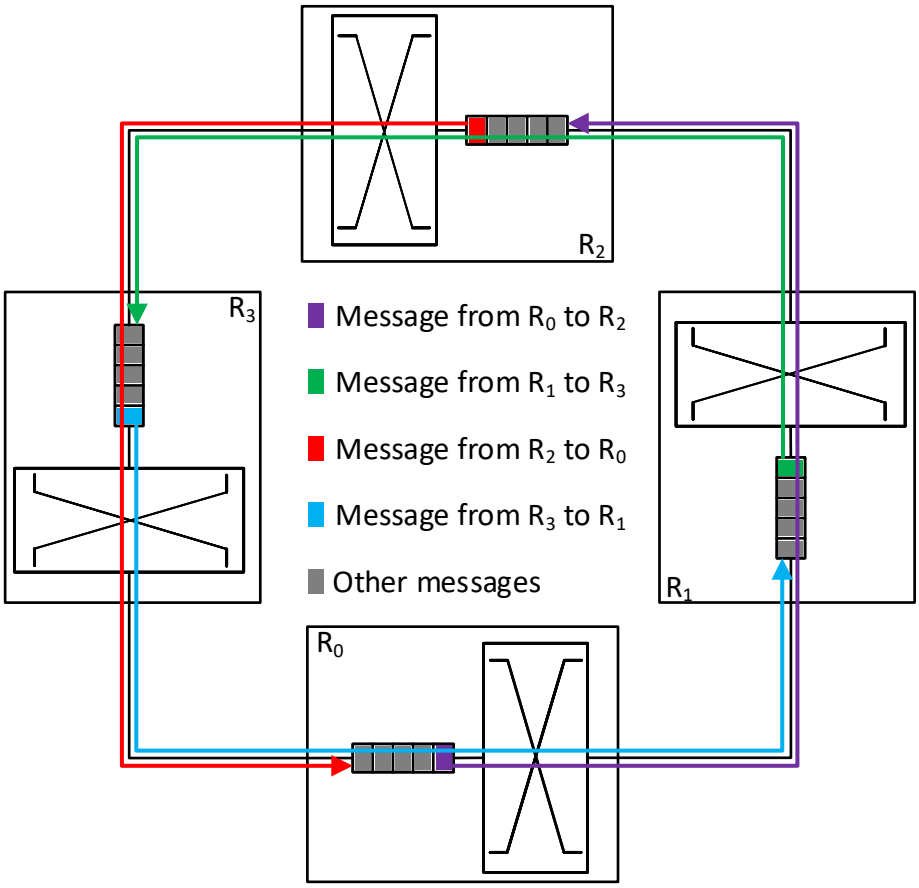


Figure 2-15: Deadlock situation due to a cyclic dependency between packets at the head of buffers in 4 routers. Each colored line represents the intended path for the packet with the same color. Note that none of the packets can advance because there is not any available slot in the next buffer of each packet, and none of the queues can drain its packets.

high-diameter networks and it was reconsidered again with the advent of low-diameter networks [110, 108, 28], is the use of virtual channels in a particular order [79]. This technique consists on splitting the physical channels into multiple virtual channels, which implies the placement of several input buffers for each network link [79, 52], as it is presented in the reference router architecture used in this work (see Figure 2-1, p. 14). There are several deadlock avoidance mechanisms based on the use of virtual channels. Günther [79] proposed that packets are stored in a different VC at each hop through the network path. Kim *et al.* [108] used a similar approach in the Dragonfly by increasing the VC index for each hop through the network path maintaining a different order for each type of network link. This implies that the quantity of VCs needed to avoid deadlocks is equal to the maximum amount of indices used for either local or global sets of VCs. The latter mechanisms is considered in this dissertation to avoid deadlocks based on VCs. This VC assignment eliminates all channel dependencies due to routing, additional VCs may be required to avoid protocol deadlock like in the Cascade system [60]. Figure 2-16 represents the same situation of Figure 2-15, breaking the cyclic dependency by the use of multiple buffers as VCs.

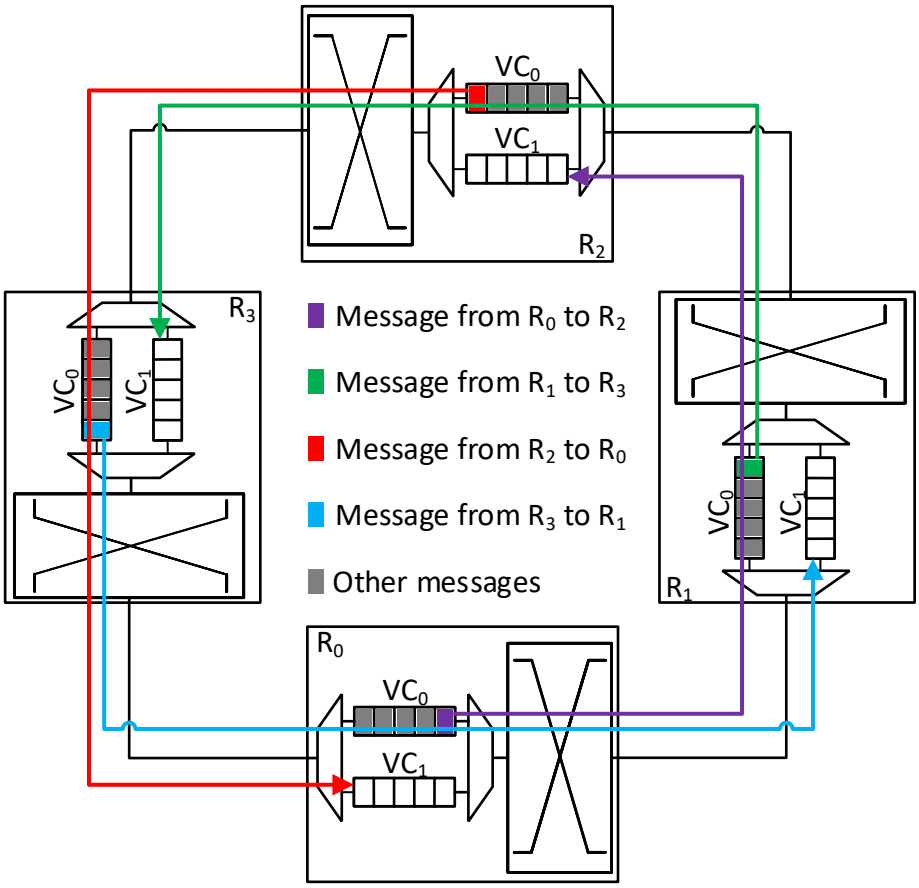


Figure 2-16: Resolution of the deadlock situation exposed in Figure 2-15 employing virtual channels to break the cyclic dependency allowing the queues to drain its packets.

Another pathological situation is *livelock*, on which packets contrary to under a deadlock, continue to move through the network and they do not reach their destinations [59]. This problem can be caused by non-minimal routing algorithms if there is not a limit of the number of times that a packet can be misrouted. A simple mechanism to solve this situation is to include a misroute counter, which holds the number of times a packet has been misrouted, into the header of the packets. Once the counter reaches a threshold, the packet must be routed minimally to its destination.

2.6 Software-Defined Networking (SDN)

To enable the network automatization and programmability, and following the same trends which have motivated the development of cloud computing, the concept of *Software-Defined Networking* (SDN) appeared. SDN is a network architecture approach which aims to make networks agile and flexible, giving network engineers programmatic control over their interconnection networks. This control is done by a central controller programmed through software applications. Hence, regarding the different layer of routers presented in Section 2.1,

the SDN architecture physically decouples the data and control planes of network devices to simplify network management. Furthermore, SDN intends to control the whole forwarding components of a network through a single control plane known as SDN *controller*, which can be replicated to support fault tolerance.

The most extended SDN standard is OpenFlow [129], defined by the Open Network Foundation in 2008 based on the concepts presented by Casado *et al.* [39]. In that work, the three principal concepts of OpenFlow were introduced:

- the central controller;
- a set of compatible switches[14];
- the concept of *flow*.

OpenFlow originally defined how the control and data plane elements would be physically separated and communicate with each other through the OpenFlow protocol, which defines the interface (*southbound API*) between them. However, it was not defined how the *business APPs* should communicate with the controller, which was denoted as *northbound API*. The controller manages switches through a set of rules or *flow entries* organized in *flow tables* inside the switches. All packets processed by an OpenFlow network device are matched against its flow entries, which can be stored in a unique flow table or in multiple pipelined flow tables, based on prioritization. OpenFlow assumes that the network devices are based on one or several TCAM matchers, so the packets match with its flows by exact match, with the possibility of using wildcards, on packet header fields.

OpenFlow's flow entry basically consists of *match fields*, its *priority*, some *statistics* and a set of *actions* to apply to matching packets [145]. Figure 2-17 depicts the above-mentioned fields and the fields from packets that a first generation OpenFlow switch is able to match [129]. Each flow entry has associated zero or more actions that dictate how the switch handles matching packets. OpenFlow compatible network devices may not support all action types described in the specification [145]. A basic subset of the required actions are.

1. Forward matching packets to physical ports and to some virtual ones [145]. This is the action equivalent to legacy routers and allows packets to be routed through the network.

2. Encapsulate and forward matching packets to the controller. This action is triggered by the first packet received of a new flow, so the controller can analyze it and decide if the flow should be added to the network device.

3. Drop matching packets is done if a flow-entry does not specify any action.

[14]Note that an OpenFlow Switch, by definition, is any network device which manipulates traffic at levels 2, 3 or 4 of the OSI model.

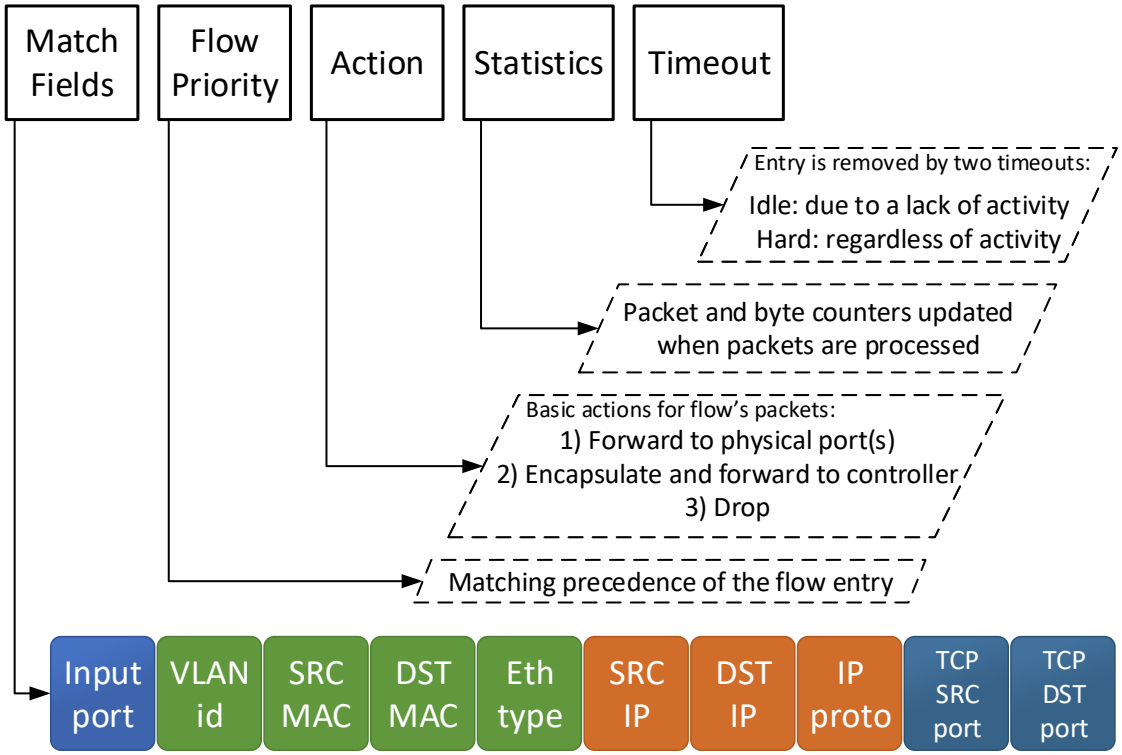


Figure 2-17: Fields of an OpenFlow's flow entry and their description.

The controller adds flow entries in the flow tables of network devices in response to packets based on the second action. Hence, the instantiation of flow entries in the network devices is reactive.[15]

OpenFlow standard has evolved quickly and has grown increasingly. However it is limited in its ability to support customized protocols and also, the process of implementing a new protocol is so long. To overcome this problem, a new packet processing language, called *P4*, was introduced by Bosshart *et al.* [34]. P4 was defined, directly, as a language to program the data plane of network devices. It raises the level of abstraction for programming the network and enables network engineers to specify their own packet headers and processing logic. Hence, the controller is not yet constrained by a fixed switch design and can control the data plane of switches. This change allows the data plane to understand and process customized packet protocols and operates inbound packets as it has been designed by the network engineers.

A general overview of the networking management evolution and the relationship between OpenFlow and P4 is depicted in Figure 2-18. In the legacy model, network devices have a fixed pipeline which supports the protocols programmed in their ASICs[16] by its vendor. In the OpenFlow model, there is an OpenFlow controller which has a complete view of the network and controls the switches. However, the hardware of these network devices con-

[15] The OpenFlow specification has evolved and currently the controller can add, update, and delete flow entries in network devices, both reactively and proactively. [146]

[16] *Application-Specific Integrated Circuit* (ASIC).

tinues to be fixed. This has been overcome in the P4 model by the advent of programmable devices. P4 programs do not specify the behavior of the control plane, however, these can be used along with the *P4Runtime* to define an interface between the control and data planes. Both options for the southbound API[17] of SDN are represented in the P4 network device on the mentioned figure.

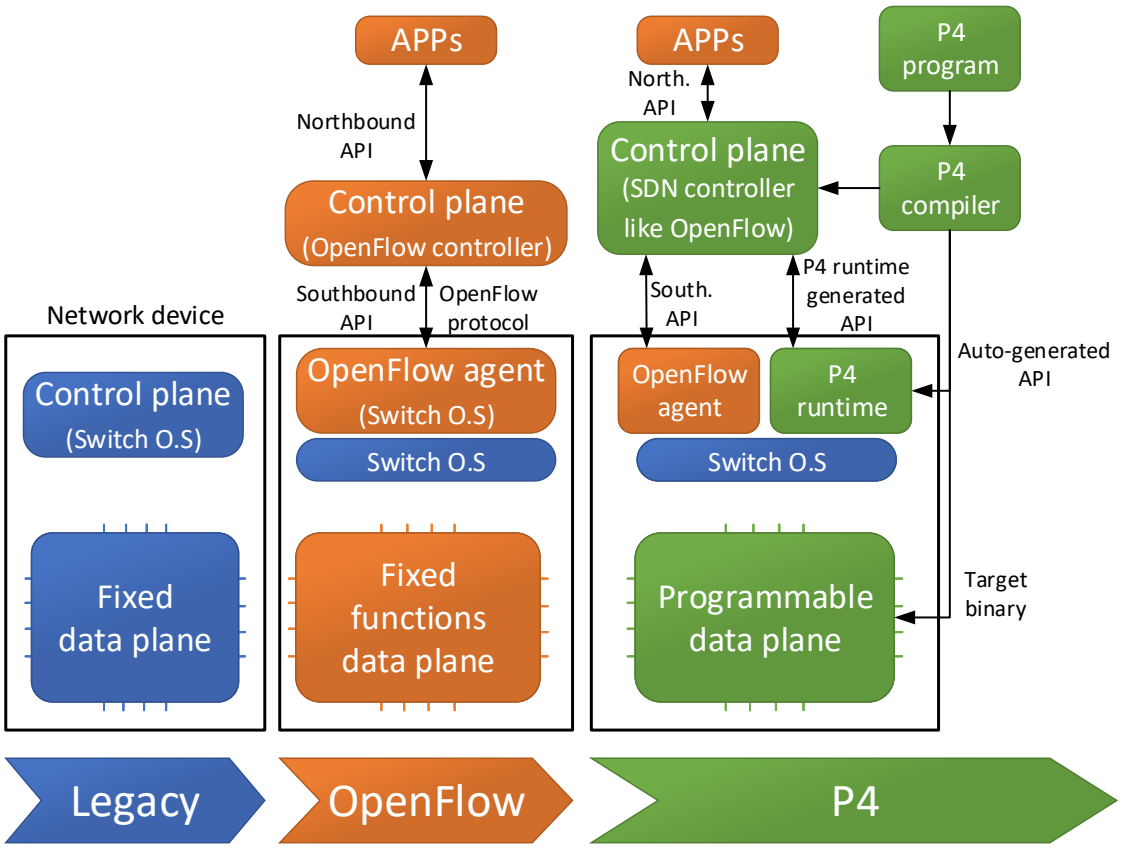


Figure 2-18: Network management evolution and the relationship between OpenFlow and P4.

[17] *Application Programming Interface* (API).

3 Methodology

All the proposals explained in this dissertation have been evaluated using the methodology discussed in this chapter. The presented methodology is common to all the chapters of this work that introduce a proposal.

The metrics used to evaluate and compare the proposals with state-of-the-art are detailed in Section 3.1. Developing and manufacturing prototypes to evaluate new networking proposals requires a lot of time and it is costly, so it is reserved to the final stages of development. However, at the early stages the software network simulators are the most useful tools, as it is explained in Section 3.2. The most affordable workload to evaluate medium to large interconnection systems are the synthetic traffic models.

The results exposed in this dissertation have been obtained with highly specialized tools, mainly by a network simulator detailed in Section 3.2.1. Section 3.2.2 describes the synthetic workloads employed to feed the network during the carried out experiments with the configuration details presented in Section 3.2.3 and summarized in Table 3-1.

In addition to the information provided in this chapter, subsequent chapters may add their own methodology section, devoted to the particular details of their experiments.

Chapter contents

3.1 Metrics

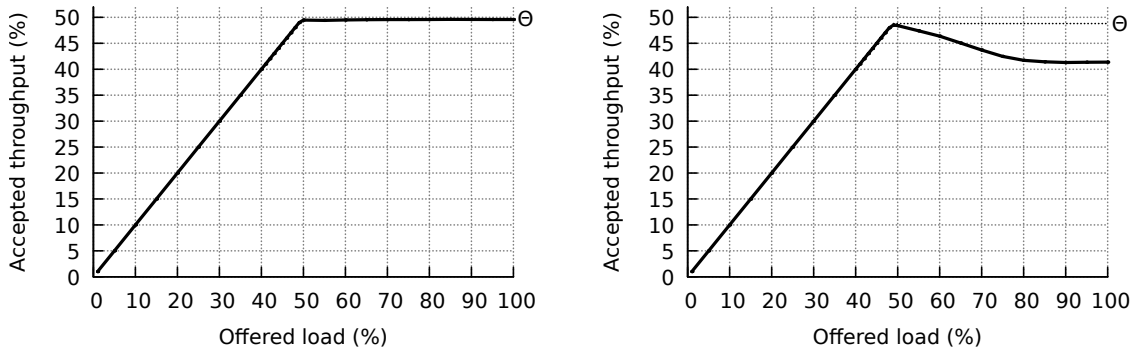
Regarding the performance of applications in a computing system, the main goal is to reduce its *total execution time*. In order to achieve that, the underlying interconnection network should transfer the maximum amount of information within the least time possible and must not to bottleneck the system. To characterize the performance of interconnection networks, a steady state is typically assumed, where there is a constant flow of messages which resembles a typical situation with multiple applications running concurrently. In this scenario, there are two basic metrics to characterize the performance of a network: *throughput* and *latency* [54]. Moreover, another important aspect to evaluate is the equity on the allocation of the network resources to terminals. This aspect is shown on the *fairness* metric.

3.1.1 Throughput

Throughput, or *accepted load*, measures the sustained data transfer rate that is effectively achieved, that is, the amount of information that is delivered over a time interval. It is measured by computing the ratio of offered load that arrives to its destination over a time interval. The typical way of representing throughput is as a function of *offered load* sweeping this from zero to one or from 0% to 100%. The throughput measured in a channel is normalized to the channel bandwidth instead of expressing it in bits per second. So, both offered and accepted load are expressed as a fraction of channel capacity or representing this as a percentage.

Throughput versus offered load curves start from very low load where throughput equals the offered load and the curve is a straight line. As offered load increases, the throughput continues straight up to *saturation point*. This is the highest value of offered load for which throughput equals to that. As offered load is increased beyond that point, the network is not capable of deliver packets as fast as they are being created and two possible network behaviors can appear. A *stable* network continues delivering the peak throughput beyond saturation point whereas an *unstable* network suffers a throughput drop, typically known as congestion, beyond that point [54]. An example of the throughput typical shape under both situations presented above is depicted in Figure 3-1.

High throughput is an important requirement to any interconnection network, which is vital in data intensive parallel applications because the amount of information that is shared between their different tasks is big. Moreover, many works that evaluate the performance of a network only show latency, because they argue that the saturation point at which the latency raises to the infinite is the throughput obtained in that network. However, showing only latency hides situations like an unstable network. In fact, this happens exactly with Figure 3-2 which indicates a throughput of 48.8% and as shown in Figure 3-1(b), above saturation point the throughput falls down up to $\approx 41\%$.



(a) Stable (b) Unstable

Figure 3-1: Curves representing throughput in (a) stable and (b) unstable networks.

3.1.2 Latency

Latency measures the time required for a packet to traverse the network from its source to its destination, that is, the elapsed time between its generation at the source and its consumption at the destination. This packet latency is made of *injection* and *network* latencies. The former is defined as the time elapsed from packet generation to its injection into the network. The latter is defined as the delay from the injection of a packet into the network and its consumption at the destination terminal. For each packet, latency is measured from the time its first bit is generated to the time its last bit is consumed and then, latency is reported as the average over all packets. The typical way of representing latency is as a function of *offered load* sweeping this from zero to saturation throughput. With non-implementable boundless injection buffers, latency beyond that point is also infinite.

Latency versus offered load curves start at the horizontal asymptote of *zero-load latency* (Ω), which can be obtained at very low offered load and represents a lower bound for the network latency. As offered load increases, latency slopes upward due to contention

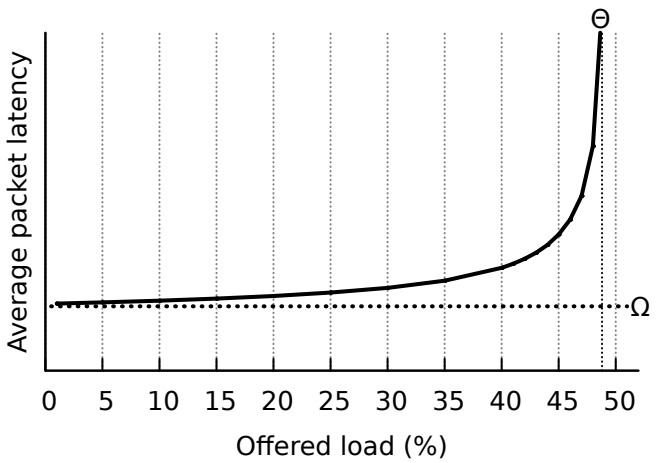


Figure 3-2: Latency curve that marks a throughput of $\approx 49\%$.

with other packets over shared resources, such as buffers and channels, because packets must wait to access them. As offered load approaches the saturation throughput, the latency curve grows to infinity as it approaches to vertical asymptote of saturation point (Θ). An example of this typical shape is represented in Figures 3-2 and 1-3(b).

Low latency is an important requirement to any interconnection network, which is vital in HPC environments, since its workloads often consist of long phases of computation interleaved with small communication ones iteratively. Hence, a high latency value on the communication phase postpones the following computation one.

3.1.3 Fairness

Fairness is a measure of the equity of the network for all terminals. In other words, in a fair network each terminal should be able to obtain an equal fraction of the network resources. It is common to analyze the *throughput fairness* which evaluates if an equal bandwidth is provided to flows competing for the same resource. There are multiple ways to measure the throughput *fairness* or *unfairness*, and this work considers a simple alternative which measures and compares the amount of packets injected per each terminal. This comparison allows to analyze any possible difference in the allocation of resources within the network or indeed within the router. The typical way of representing throughput fairness is plotting the quantity of packets injected by each terminal, or grouping them by the hosts of each router, in a vertical bar which can range from zero up to the employed offered load, typically below saturation point.

A throughput fairness situation is represented by a set of bars reaching approximately the same injected load value whereas an unfairness situation is easily detected when the injected load by a router or set of routers is significantly lower or higher than by the others. An example of the typical shape of a plot under both described situations is represented in Figure 3-3.

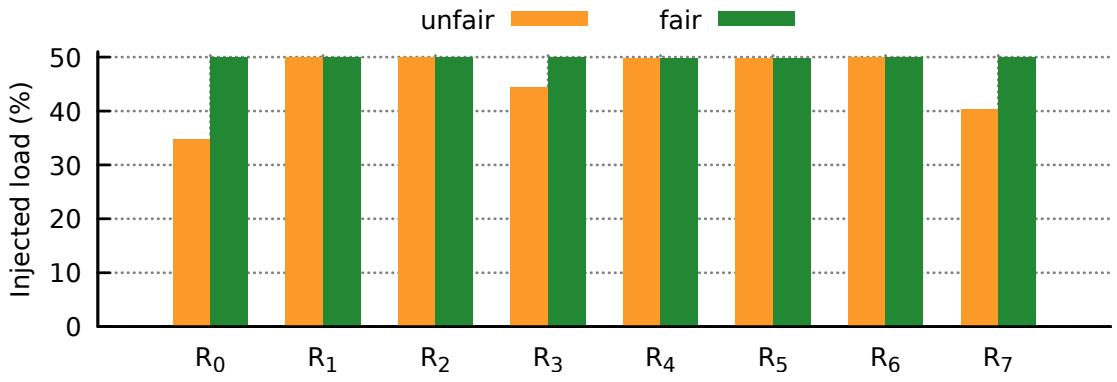


Figure 3-3: Typical throughput fairness shape with a fair case in green and with an unfairness problem in routers 0, 3 and 7 in orange. The offered load is 50%.

The unfairness problem exposed in Figure 3-3 in orange can be detected additionally in a throughput plot when the accepted load before saturation point is not equal to the offered load, as can be seen in Figure 3-4. This can be observed by looking at the gradient of the throughput curve in detail because is lower than the expected 45°.

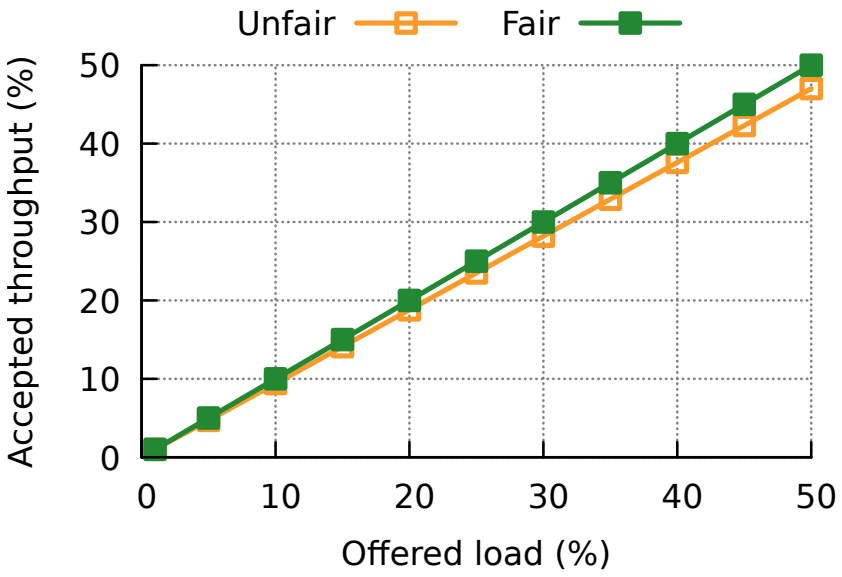


Figure 3-4: Throughput fairness representation using throughput curves for a fair situation in green and showing an unfairness problem in the orange curve.

3.2 Simulation

Computer engineers need tools for the design and evaluation of new systems. Regarding the network, developing and manufacturing prototypes requires a lot of time and it is costly, so that is why it is usually postponed to the final stages of a system development. Even though analytical tools that can model many aspects of a network exist, there are situations too complex to represent with these tools. Hence, in these cases and in early stages of a system development, software network simulators are the most useful tools. They simulate an interconnection network confronting a certain network workload to extract different network metrics.

There are different levels of detail for network simulators [54], which are summarized next from the highest to the lowest level of detail. Interconnection networks can be simulated at *Register-Transfer Level* (RTL) to fine-tuning the router internals but this approach is extremely time-consuming. *Cycle-accurate* simulators have a precise representation of the router internals and reduce greatly the simulation time over RTL simulations. However, their answer time is yet big when it is required to simulate a large interconnection network. Finally, the *functional* simulators simplify the router internals to reduce the computational requirements and their simulation time. This work has employed cycle-accurate intercon-

nection network simulators because they have enough accuracy for the type of experiments developed and the computational requirements and simulation time of those are affordable.

There are different levels of precision for network workloads or the traffic pattern applied by the terminals [54]. Ideally, the experiments should be done with the most precise workload, which is derived from actual execution of applications, that is, *application-driven* workloads. There are two possible application-driven workloads: *execution-driven* and *trace-driven* workload. Whereas on the former, packets are generated by a certain application running over simulated terminals, on the latter, the sequence of packets sent by each terminal running that application are captured and later replayed by simulated terminals. Both methods are employed in some cases. However, experiments using this kind of workloads typically require a big amount of computational requirements and a lot of time. Moreover, the previous techniques are not practical for medium to large simulated interconnection networks. Another option is *synthetic* workloads, which can capture the demands of an application to the interconnection network if they are carefully designed. On these workloads, each terminal typically injects packets according to a *Bernoulli process* with variable load for a certain *traffic pattern* considered. This dissertation employs this latter type of network workload for its experiments, which allows to evaluate its proposals providing enough insight about the network performance in a reasonable time using the different traffic patterns presented in the next section.

An important aspect of software network simulators which impacts on its design and answer time, mainly at low and medium loads, is how the simulation is conducted. Simulators can be either *time-driven* or *event-driven*. In the former, the simulation of each actor, such as terminals or routers, is triggered by an increment in the time cycle counter. The triggering of an actor is done regardless of that actor has any effective outcome at the current cycle. In the latter, the events generate other events to be performed by the same or other actor of the network. Hence, the triggering of an actor is solely done when there is an event for it instead of every cycle like in time-driven.

3.2.1 FOGSim interconnection network simulator

Unless otherwise stated, the results presented in this dissertation have been obtained employing the open-source *FOGSim network simulator* [73]. FOGSim is a *phit-level*, *cycle-accurate* and *time-driven* software network simulator, which supports a range of several synthetic traffic patterns and traces obtained from parallel applications in the Dimemas [117] format. It has been employed in several works to evaluate proposals for HPC interconnection networks.

Abstractly, FOGSim simulates the network as a collection of routers and channels in a canonical Dragonfly and wires the global network links following the *palmtree* global link arrangement explained in Section 2.2.1. Several routing mechanisms are implemented in

the simulator, including all the ones presented in Section 2.3 and all the proposals detailed in this dissertation. A credit-based flow control is employed for buffer management between adjacent routers and a dedicated network link is used to communicate credit information to upstream routers. Deadlock is prevented using the amount of required *Virtual Channels* (VCs) and increasing the VC index in each hop, following the idea presented in [108].

Considering the level of detail, FOGSim simulates the network at the granularity of phits and clock cycles. A packet, which contains routing and sequencing information, is broken into one or more flits. Its size is expressed in phits, which is the unit at which the data is transferred physically through the network. On each clock cycle, a network channel can read and write respectively a single phit from its input and to its output. For modeling the network delays, arbitrary delays can be assigned to the crossbar of the router and to the network channels, varying by their type, local or global.

Simulated routers employ virtual cut-through switching and can be either input-queued or *Combined Input-Output Queued* (CIOQ). The size for each type of buffer can be set independently. An input-first separable allocator, according to the explanation on Section 2.1, is employed to assign the router resources. There are different policies that can be applied to the arbiters of the allocator but the default behavior is a RR arbitration in which the priority list of ports is updated only when the arbiter generates a winning grant. The simulator allows to set an internal speed-up, which increases the crossbar operation speed in comparison to the network channels. In this case, a CIOQ router queuing organization is mandatory, because the output buffers decouple the crossbar and channel speeds.

During network initialization, elemental network characteristics are automatically derived based on the input parameters, such as network size, the amount of routers and network links, and the routers' radix. All routers are configured identically regarding the router architecture, routing and flow control; this is based on the input parameters detailed in Section 3.2.3. The simulated routing computation control unit has enough access to the flow-control information to be able to implement adaptive routing algorithms based on it, such as output port credits.

3.2.2 Synthetic workloads

A perfect routing algorithm for a particular traffic pattern, which almost achieves the performance bound imposed by the topology, may have, however, a poor performance on other traffic patterns. For example, minimal routing provides the best performance under a uniform spatial distribution of the packets, whereas the performance obtained with that routing under adversarial traffic patterns is appalling. Hence, network engineers need to ensure the performance results of an interconnection network in a wide range of traffic patterns.

This section presents the traffic patterns employed to evaluate the proposals presented in this dissertation. It details the definition, special features and a real-world example of

every presented traffic pattern. First, Section 3.2.2.1 presents *steady-state* traffic patterns and secondly, the *transient* traffic pattern is presented in Section 3.2.2.2.

Even though the above-mentioned traffic patterns can happen in practical systems, traffic patterns in real world interconnection networks depend on many factors such as compute nodes allocation made by the job scheduler [93, 197] and the logical traffic patterns produced by the applications [29].

3.2.2.1 Steady-state traffic patterns

Steady-state traffic patterns resemble the typical situation in a network where there is a constant flow of messages. This situation occurs when there are multiple applications running concurrently on different terminals connected to the interconnection network. The traffic patterns detailed next are a representation of *benign* and *adversarial* traffic patterns. The former naturally balance load in the network whereas the latter cause load imbalance in the links of the interconnection network.

3.2.2.1.1 Random Uniform (UN) Under *random Uniform* traffic pattern (UN), usually denoted as *uniform*, the probability of sending a packet to each destination is equal. Hence, the target of each packet is any randomly selected compute host among all terminals in the interconnection network, except the source compute node.

This traffic pattern is *benign* for the interconnection network. In fact, it is the best traffic pattern to confront by a Dragonfly network, since it balances the use of interconnection network channels. This property is not only applicable to Dragonfly topology and, so, this traffic pattern is commonly employed during the evaluation of interconnection networks to obtain performance measurements under a *best-case* scenario [54].

Under UN traffic pattern, the optimal routing algorithm to apply is the minimal routing because the load under UN is already spatially distributed, and employing the shortest path achieves an optimal latency result.

3.2.2.1.2 Adversarial shift (ADV+*i*) Under *Adversarial shift* traffic pattern (ADV+i), usually denoted as *adversarial*, the destination of a packet is selected randomly from all nodes within the group located i groups ahead of the source modulo the number of groups. The only global link between source and destination groups, departing from a router denoted as R_{OUT}, becomes the bottleneck under minimal routing. In a balanced Dragonfly network, it restricts throughput to $\frac{1}{a \cdot p} = \frac{1}{2h^2}$ phits/node/cycle using MIN routing. Figure 3-5 represent this traffic pattern over a Dragonfly network, including the two variants of them described next.

Many works that study a Dragonfly network have employed adversarial shift traffic with an offset $i = 1$. However, other offsets can be employed leading to multiple variants of this traffic pattern depending on the value assigned to i. Furthermore, the adverseness of each ADV+i

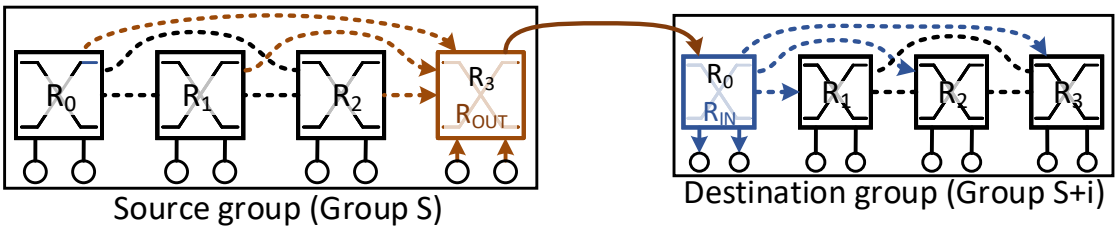


Figure 3-5: Representation of the ADV+*i* traffic pattern in a Dragonfly network.

traffic pattern is different fore some routings. Particularly, the variant obtained when $i = h$, where h is the number of global network channels per router, is very relevant. This variant is a pathological case because it not only stresses the global network channel between source and destination group. Indeed, when the global wiring is consecutive, all the traffic received in an intermediate router selected by the Valiant routing algorithm should be routed by the h global links in a subsequent router, stressing the single local link between both routers. This particular problem is highlighted in Figure 3-6 and it will be revisited throughout the following chapters.

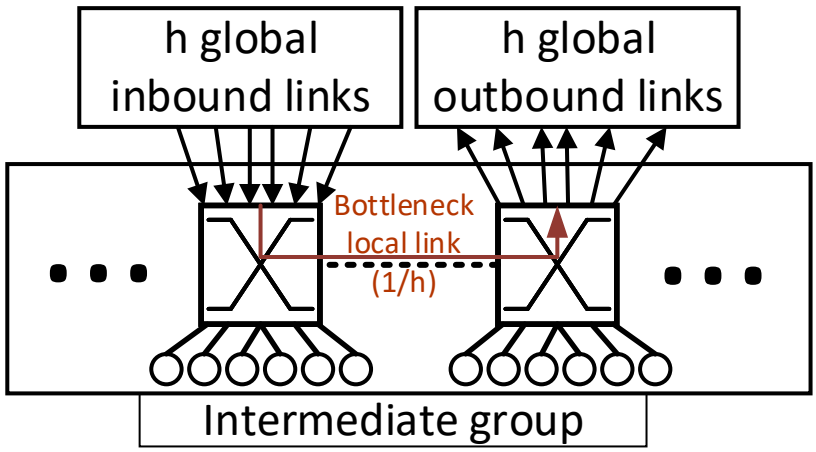


Figure 3-6: Representation of the bottleneck at local link of intermediate group under ADV+*h* traffic pattern in a Dragonfly topology with *h*=6.

This ADV+i traffic pattern is adversarial for the interconnection network; in fact, it is the worst traffic pattern to confront by a Dragonfly network. Under minimal routing, the bottleneck generated enforces the need for a non-minimal routing to deal with it and obtain a good performance. Oblivious Valiant routing, explained in Section 2.3.2, randomizes traffic and avoid bottlenecks, then, it can be considered the reference to beat under this traffic pattern. However, the additional hops of its *phase A*, where packets are diverted to R_{ROOT}, double the path length (lgl – lgl) and increase base latency.

This traffic pattern can occur in a network executing real-world applications. For example, ADV+1 traffic pattern is present in an MPI collective communication that uses the ring algorithm [178] when the nodes dedicated to the application are allocated adjacently. An-

other case is a multidimensional stencil pattern when the allocation of nodes is sequential and the offset in any dimension is bigger than the group size. Jain *et al.* [93] consider the effect of job placement on a Dragonfly network for a 4-D stencil.

3.2.2.1.3 Adversarial Local (ADVL) Under *Adversarial Local* traffic pattern (ADVL), the destination of a packet is selected randomly from all nodes in the consecutive router within the same group, following a modulo sequence. It concentrates all minimally routed traffic on a single network channel between the two routers, restricting throughput to $\frac{1}{p}$ phits/node/cycle using minimal routing. Figure 3-7 represents this traffic pattern over a Dragonfly network.

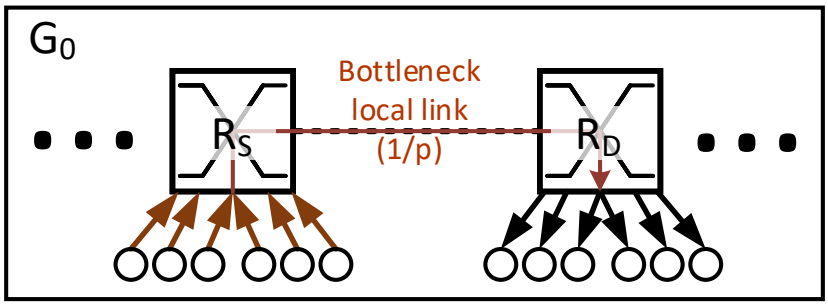


Figure 3-7: Representation of the ADVL traffic pattern and the bottleneck in the local link between source and destination routers.

To cope with this traffic pattern, a non-minimal routing is required. However, the same routing which applies to ADV+*i* traffic pattern may not apply directly to them because it sends messages between pairs of terminals which belong to the same group. This idea will be developed in Chapter 5.

Like the previous traffic pattern, ADVL traffic pattern can occur in the same type of communication if the number of allocated nodes is lower than before, and they are also allocated adjacently, so the communication is local within a group. This traffic pattern has been observed with stencil workloads by Kerbyson *et al.* [104].

3.2.2.1.4 Adversarial Consecutive (ADVC) Under *Adversarial Consecutive* traffic pattern (ADVC), the destination of a packet is selected randomly from all nodes in h destination groups. Specifically, packets are sent to the h consecutive groups $(+1, +2, ..., +h$ modulo $G = 2h^2 + 1)$ after the source group, which are all connected to the same bottleneck router within the source group. Figure 3-8 represents this traffic pattern over a Dragonfly network.

This traffic pattern could be considered adversarial for the interconnection network, not as much by throughput limitation, which is less than ADV+*i* and equal to $\frac{h}{a \cdot p} = \frac{1}{2h}$ phits/node/cycle using MIN routing. However, this traffic pattern represents a challenge

in terms of throughput fairness, since the bottleneck router represented as R_{OUT} in Figure 3-8 gets all its global links occupied by the traffic routed minimally from all other routers in the group. Hence, compute nodes in the source switch R_{OUT} and terminals in other routers of the same group observe very different local traffic conditions. At the end, this problem can lead to starvation at certain nodes if the routing algorithm does not make a proper job.

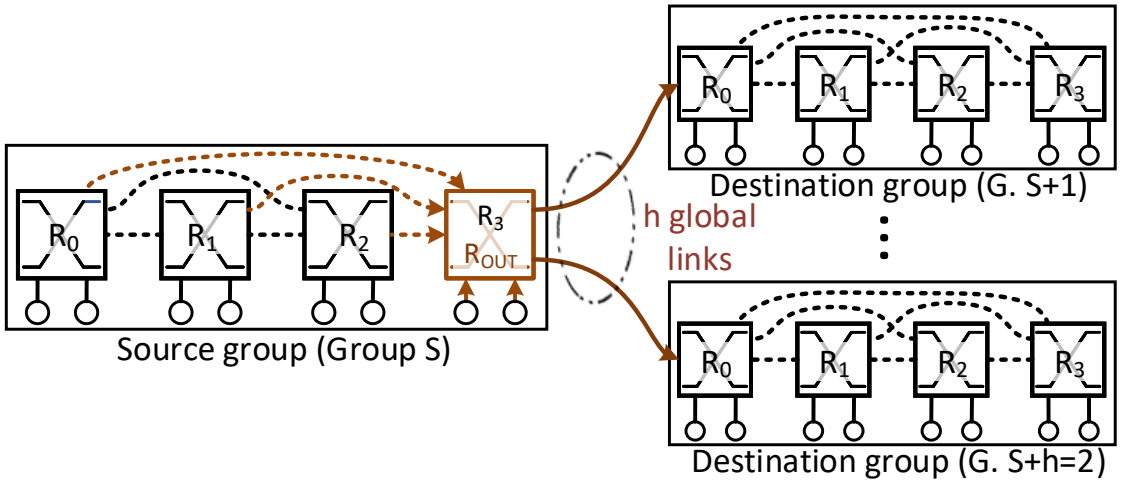


Figure 3-8: Representation of the ADVC traffic pattern highlighting the bottleneck in global links.

ADVC can occur in the ring communication example presented for ADV+1 if the consecutive allocation of the nodes dedicated to an application takes up to *h* groups. In this case, uniform traffic between the application processes becomes an ADVC traffic pattern from the first group point of view.

3.2.2.1.5 Mixed (MIX) In addition to a static situation on which all the terminals of the network inject the same traffic pattern, the terminals may inject different patterns at the same time. To be able to evaluate how a network deals with a combination of traffics, the *Mixed* traffic pattern (MIX) is introduced.

This traffic pattern is able to offer a given percentage of different traffic patterns for a required injection load. For example, if the network should be loaded with a 50% of its capacity mixing UN and ADV+1 evenly, each of the terminals in the network injects 25% of its capacity following UN traffic and another 25% using ADV+1.

3.2.2.2 Transient traffic pattern

In addition to the steady-state loads presented above, the networks manage different traffic patterns alternatively. To be able to evaluate the transitions between those traffic patterns, this *transient* traffic pattern is introduced.

This traffic pattern is able to offer load following one specific steady-state traffic pattern and in a discrete time, denoted as time $t = 0$, change it to another. In this case, the network

is warmed-up with the traffic pattern assigned to the first phase but the results begin to be measured after the amount of cycles setted as warm-up. During the whole simulation different metrics, such as the packet latency and the number of misrouted packets are recorded. Moreover, the traffic pattern is able to apply the same steady-state traffic pattern with a given offered load on each of its phases.

This traffic pattern allows to determine how quickly a routing algorithm reacts to load changes, i.e., it measures the response time of a routing algorithm. And not only between different traffics, it is also possible to evaluate the response to changes in the offered load.

3.2.3 Simulator configuration

Unless otherwise stated in the particular methodology sections of the following chapters, the experiments carried out to obtain the results presented in this dissertation employ the simulation parameters explained below and summarized in Table 3-1.

The experiments model a Dragonfly network with $p = h = 6$ compute hosts and global links per switch respectively and $a = 12$ routers per group. The simulated network, which is not extrapolated from a single specific system but it is realistic and representative of current state-of-the art HPC systems, can be built using 24-port routers and it comprises more than 5K compute hosts and almost 1K routers grouped in 73 groups.

The developed network simulator models, depending on the case, an input-queued or combined input-output queued routers (see Figure 2-2, p. 15) using virtual cut-through switching and operating at 1 GHz with round-robin policy for both input and output arbiters of the allocator. Also, the minimum number of virtual channels required have been employed to avoid routing deadlocks, incrementing the VC index in each hop, distinguishing between local and global network links [108]. For example, a VLB_{lgl} path with six hops ($l_1g_1l_2 - l_3g_2l_4$) requires only four VCs, as it is indicated by the biggest subindex of each type of hop. For injection ports no one virtual channels have been considered.

The technology parameters employed in the experiments mimic an HPC switch with a 90 ns port-to-port latency. The network links bandwidth and latencies, and the packet size are customized in the following chapters because Chapter 4 focuses on HPC networking over Ethernet technology whereas Chapters 5 and 6 are agnostic of the network technology. Furthermore, technology has evolved during the development of this PhD work. For example, due to state-of-the-art networking link speeds have been used, the link speed employed in the evaluations has increased from 40 to 200 Gpbs. The size of router buffers is selected to cover the round-trip time to support lossless flow control. An internal router speed-up of 2x over network links is employed.

Each point of the evaluation results has been obtained averaging a battery of ten simulations on which network statistics were collected for 60,000 cycles after an identical length network warm-up.

Table 3-1: Simulation parameters employed in the experiments.

	Parameter	Value
Network configuration	Topology	Dragonfly with $h = 6$
	System size	$N = N_{MAX} = 5,256$ compute hosts
	Number of groups	$G = 73$ groups
	Switches per group	$a = 12$ routers
	Switch degree	$k = 23$ ports
	Global link arrangement	Palmtree
	Switch queuing strategy	Combined input-output queuing
	Switching mechanism	Virtual cut-through
	Arbitration policy	Round-robin
	Switch frequency	1 GHz
	Switch latency	90 ns
	Internal crossbar speed-up	2×
	Link speed	200 Gbps
	Local link latency	15 ns (3 m)
	Global link latency	150 ns (30 m)
	Packet size	10 phits, 250 bytes
	Injection queues size	126 KBytes
	Local transit queue size	18 KBytes
	Global transit queue size	45 KBytes
	Output queue size	15,75 KBytes
	Number of VCs in injection ports	0
	Number of VCs in global ports	1 for MIN and 2 for other routings
	Number of VCs in local ports	2 for MIN and 4 for other routings
PB	Global link state calculation (Eq. 2-2, p. 33)	$F = 120\%$, $T = 5$ flits
	UGAL threshold constant (Eq. 2-1, p. 33)	$T = 0$ flits

HPC Networking Over Commodity Ethernet Technology

4

Exascale systems will require large networks, typically based on low-diameter and high-radix topologies such as Dragonfly, with hundreds of thousands of endpoints. These networks usually rely on non-minimal adaptive routing algorithms to deal with varying traffic characteristics and avoid pathological performance.

Ethernet technology is employed in a significant fraction of the Top500 systems, and will remain as a cost-effective alternative for *High-Performance Computing* (HPC) interconnection networks. However, its current design is not scalable to exascale systems. Different solutions have been proposed for scalable Ethernet fabrics for *Data Centers* (DCs), but not specifically for HPC applications.

This chapter identifies the major differences in network requirements from both environments in Section 4.2. Based on them, it proposes the application of Ethernet to exascale HPC systems, considering the topology, routing, forwarding table management and address assignment, with a focus on performance and power. The proposed solution for scalability, in Section 4.3, relies on OpenFlow switches to implement hierarchical addressing with a mechanism to reduce the forwarding table size. To simplify deployment, Section 4.4 introduces a protocol that performs automated address assignment without interfering with layer-2 service announcement protocols.

This chapter introduces two different adaptive routing mechanisms: *MAR-bP* and *QCN-Switch*. The first one, introduced in Section 4.5, overcomes the infeasibility of using controller-based per-flow traffic engineering introducing the *conditional flow rules* to allow for adaptive routing with proactively rule instantiation. Routing decisions are taken by switches, based on Ethernet's *pause* frame, without controller interaction. The second one, introduced in Section 4.6, updates the *conditional flow rules* associating a probability value to each output port. This value is updated, based on snooped explicit congestion notification messages, to reflect downstream congestion and used to statistically divert traffic away from congested areas when the load is uneven.

Altogether, this chapter introduces a realistic and competitive implementation of a scalable lossless Ethernet network for exascale-level HPC environments, considering low-diameter and low-power topologies such as Flattened Butterflies or Dragonflies, and allowing for power savings up to 54%. Evaluation results show that QCN-Switch is a competitive design for both uniform and adversarial traffic. Furthermore, it is able to react to changes in traffic conditions in 0.4 ms or less. A sensitivity analysis identifies the best configuration and shows its performance trade-offs.

Chapter contents

4.1 Motivation

Technology evolution has led to a convergence in DC and HPC systems. In fact, similarly to the introduction of commodity x86 processors in HPC systems in the 1990s - 2000s, nowadays a significant part of the HPC systems rely on commodity Ethernet technology. Introduced previously, the Figure 1-4 shows the evolution of Ethernet technology in the Top500 [179] list for the last three decades. At the moment, it has a significant space with a systems share close to 53% of the Top500 list.

Ethernet's large economy of scale [40], the advent of simple white-box switches [144] based on merchant silicon, the possibility of lossless implementations [88], and the ubiquity of Ethernet NICs in SoCs[1] or motherboards suggests it will remain as a cost-effective alternative for HPC interconnection. However, commodity Ethernet interconnect design is not directly scalable to exascale systems. The number of computing nodes required for such systems would be very large; with 3 TeraFLOPS nodes,[2] more than 300,000 of them would be required, clearly exceeding the capacity of the forwarding tables of any commodity switch. Scaling these tables would impact both switch latency and power consumption.

Although several technologies have emerged to scale Ethernet networks, such as overlay encapsulations or *Software-Defined Networking* (SDN) solutions based on OpenFlow [129], they are not necessarily suited for an HPC environment. In fact, the flexibility requirements in a DC differ from the low-latency and high-throughput goals in HPC systems.

Topologies proposed for massive-scale data centers or exascale systems, such as Dragonfly [108], sacrifice minimal-length path diversity for larger number of nodes and lower average distance. Minimal paths in these topologies are heavily congestion-prone, requiring the use of non-minimal paths typically by non-minimal adaptive routings [108, 96, 74]. Most of the previous proposals rely on comparing congestion indicators between two or more paths to select the most appropriate one. These congestion indicators are typically derived from buffer occupancy, either directly measured from the credit count of the link-level flow-control mechanism of its neighbor switches, or notified by the network in case of congestion in remote areas (in some form of explicit congestion notification) in addition to credit counters.

Some network technologies, such as lossless Ethernet, do not implement link-level flow-control credits,[3] which makes direct sensing of buffer occupancy unfeasible. Hence, this

[1] *System of a Chip* (SoC).

[2] There are two trends to build supercomputers: based on *fat* or *thin* nodes. An example of fat node, with a performance of ≈ 46.8 *TFLOPS*, is the IBM Power System AC922 server model 8335-GTW [31] present in the Summit system [184]. By contrast, an example of thin node, with a performance of ≈ 3.38 *TFLOPS*, is developed by Fujitsu with the A64FX processor [71] for the Fugaku system [69]. These values are peak performance for double precision.

[3] The first flow-control mechanism for Ethernet, the *pause* frame, was defined by the IEEE 802.3x standard [87]. Follow on from 802.3x, *priority flow control* 802.1Qbb [88] operates on individual priorities using pause messages per each *Class-of-Service* (CoS).

motivates the design of a non-minimal adaptive routing based on *pauses*. Moreover, even if credit counters could be used, remote explicit congestion notification messages send information about the status of remote areas of the network that might suffer congestion. Then, this information can be used to simplify source-adaptive routing algorithms. Hence, these two previous arguments together motivate the design of a non-minimal adaptive routing algorithm based *solely* on *Explicit Congestion Notifications* (ECNs).

4.2 Interconnection requirements: HPC vs. DC

Scalable SDN solutions have been proposed for large-scale DC networks based on commodity OpenFlow Ethernet switches. However, network requirements and traffic characteristics in an HPC environment differ from those in traditional DC, making the optimal solutions in one case not so suited to the other. In particular, the main requirements identified to differ in HPC from traditional DCs are the following.

- HPC communication phases are typically shorter.
- Low latency is crucial for application performance, making controller-based flow instantiation unfeasible.
- The communication stack is not necessarily TCP/IP.
- Topologies different from traditional folded-Clos have been proposed specifically for large HPC systems, with a special focus on both scalability and power saving.

The next paragraphs analyze these differences and map them to implications on the underlying network fabric.

Communication phases are much shorter in HPC applications than in DC workloads, especially in DC user-facing applications. This makes controller-based per-flow traffic engineering (adaptive routing) unfeasible. Several traces of applications from the *NAS Parallel Benchmarks* (NPB, [16]) have been analyzed, from runs with 64 MPI[4] processes using 64 nodes of a cluster. Figure 4-1 shows a visualization of four iterations of the CG kernel. Blue sections represent computation phases, orange sections represent communication ones and yellow lines are point-to-point messages. Four iterations of the algorithm last 6.48 ms,[5] resulting in changes of traffic in less than 2 ms. A similar visual analysis of other benchmark applications gives rise to the values in Table 4-1. Iterations range from 2 to 58 ms, meaning that traffic changes are even quicker. By contrast, typical DC applications can suffer from congestion periods lasting for several seconds [100].

Several proposed mechanisms rely on controller-based estimations of per-flow offered load in order to adapt routing [62, 137, 63]. However, their reaction time exceeds

[4] *Message Passing Interface* (MPI).
[5] The timescale of the trace is indicated in the bottom of the figure in small font

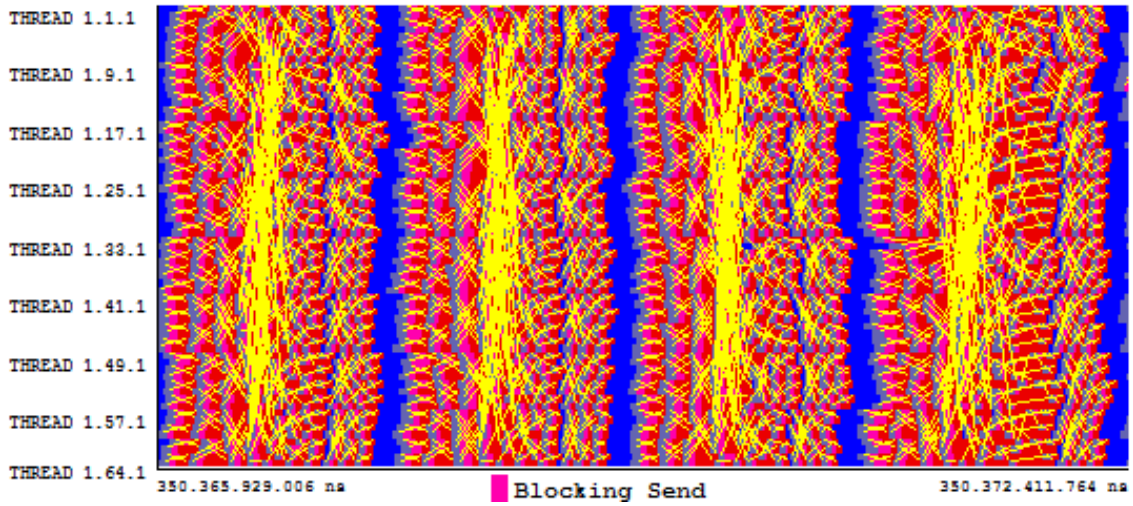


Figure 4-1: Visualization of CG kernel using 64 MPI processes. Blue and orange sections represent computation and communication phases, respectively. The yellow lines are point-to-point messages. The time frame shown comprises 6.48 ms for 4 iterations.

Table 4-1: Approximate iteration time of NPB applications.

Application/kernel	Iteration time
CG	1.6 ms
BT	29 ms
FT	58 ms
IS	10 ms
LU	52 ms
MG	11 ms
SP	22 ms

70 ms in the best case, due to monitoring intervals and remote controller communication. Planck [159] captures traffic samples to react in the order of few milliseconds. However, it requires a significant monitoring infrastructure (including sparse switch ports, collectors and controllers) and relies on TCP sequence numbers to estimate flow rates, making it specific for traditional DC environments. As far as it is known, no other monitoring mechanisms achieve reaction times similar to Planck, making controller-centric traffic engineering unfeasible in HPC environments. Therefore, reactive flow rules instantiated by the controller are not well suited to adapt switch routing to changing HPC traffic. The latter joined with low latency requirements [26] impose the need of proactive forwarding rules, rather than reactive ones instantiated when a flow begins, to avoid traffic detours in the critical path. An SDN controller should learn the network topology and set the forwarding rules in advance,

like the *subnet manager* in InfiniBand [153] networks. Of course, network flooding[6] due to missing forwarding entries should be avoided.

HPC alternatives to the TCP/IP communication stack, such as Open-MX [77] and *RDMA*[7] *over Converged Ethernet* (RoCE, [92, 130]), do not employ IP addresses nor TCP ports.[8] This makes a single layer-2 domain compulsory. A traditional Ethernet design with *flat* addressing in an exascale system would require *Content-Addressable Memory* (CAM) forwarding tables with hundreds of thousands of MAC entries. That is impractical with traditional switch hardware not only because of table capacity limitations, which typically ranges from 4K to 64K entries [44, 46], but also of power and latency concerns. In fact, there are switches which can reduce their latency by reducing their effective CAM size [46].

DC interconnection networks traditionally rely on some form of tree or folded-Clos topology. Implementation costs and energy consumption restrictions in HPC environments suggest the use of alternative, direct topologies with lower diameter, such as Flattened Butterflies or Dragonflies. Compared to folded-Clos, these topologies employ a lower number of switches and links for a given network size, leading to a lower power consumption in switches logic and link *Serializer/Deserializer* (SerDes), and lower installation cost. However, contrary to Clos-like topologies, minimal paths in these topologies are heavily congestion-prone, requiring *non-minimal* adaptive routing as discussed in Section 2.2.

A network design should suffice all the discussed requirements. The next section presents an analysis of different solutions which have been considered for DC environments and discusses their applicability to HPC, starting from the layer-2 scalability issues.

4.3 Scalability mechanisms in Ethernet networks

Traditional Ethernet employs flat routing based on the hosts MAC addresses. Switches' CAM forwarding tables employ an entry for each endpoint in the network. Conversational MAC address learning in traditional Ethernet switches [1] fills these tables dynamically when network conversations start, avoiding allocating entries for destinations without communication. However, this reactive mechanism implies that frames destined to unknown entries are flooded across the whole network to make sure they reach their destination. Indeed, a major disadvantage of a flat addressing is the required size of the switches' MAC address table: when it overflows, traffic to MACs in excess are flooded as if they were unknown destinations.

Overlay mechanisms have been proposed to build large layer-2 fabrics. *Transparent Interconnection of Lots of Links* (TRILL, [151]) and Cisco's FabricPath [45] employ MAC-in-MAC encapsulations to reduce the forwarding table use. In these hierarchical implementations,

[6] Used in traditional Ethernet with conversational learning, 802.1D [1].
[7] *Remote Direct Memory Access* (RDMA).
[8] RoCEv2 changes the packet encapsulation to include IP and UDP headers which allows to use RDMA across both L-2 and L-3 networks [132].

ingress switches encapsulate the original Ethernet frame in a new frame using the destination switch ID for the destination MAC field; egress switches remove the outer header to recover the original frame. Switches in the *core* of these networks only need to hold CAM entries for the network switches, not the network hosts. By contrast, *access* switches still require an entry for each host involved in a conversation. This could explode to the whole network size in the worst case. Conversational learning is also employed by default in these proposals, what implies that unknown (or overflown) entries are flooded.[9] *Virtual extensible Local Area Network* (VXLAN, [124]), which is a MAC-over-UDP/IP overlay encapsulation, also relies on conversational learning and does not reduce the maximum size of access switch tables.

These previous approaches rely on searches on CAM tables for a match of the complete MAC address. By contrast, OpenFlow switches employ *Ternary-CAM* (TCAM) tables allowing for partial matches and hierarchical MAC organizations. PortLand [143] and MOOSE [163] introduce layer-2 hierarchical routing, by organizing hosts in groups sharing a common MAC prefix and setting a single routing rule for all hosts in the same destination group. A unique *pseudo-MAC* (PMAC) address is assigned to each host, encoding its location in the network. Access switches dynamically modify the MAC field of the frames received from or sent to its own nodes (MAC to PMAC and PMAC to MAC), so the network only detects PMACs and destination hosts maintain the illusion of unmodified MAC. Additionally, these access switches divert *Address Resolution Protocol* (ARP) petitions to the network controller which responds with the destination's PMAC.

Like the latter studied proposal, the proposed scalability in this chapter relies on hierarchical MAC addresses, since they require a limited number of proactive forwarding rules and do not flood traffic. This layer-2 hierarchical MAC addressing is adapted to the employed topologies, which have been proposed for larger HPC environments. The number of flow entries depends on the definition of groups. Section 4.3.1 analyzes the rules required for different address hierarchies and topologies, and Section 4.3.2 studies rule compaction.

4.3.1 Scalability analysis of hierarchical addressing

The size of the routers' TCAM table determines its power consumption and could limit the scalability of a layer-2 OpenFlow network fabric. Hence, the objective of this proposal is to reduce the number of flow rules required to maintain the communication between terminals of an HPC system and this also helps to reduce its power consumption. To achieve that, a layer-2 hierarchical addressing organization is designed, in which addresses are divided into different *blocks*. An address block can be represented by a wildcard when the information represented by this block is not relevant. This section explores the number of flow entries

[9] The ESADI protocol [196] implements proactive address learning in TRILL, but it still requires CAM tables with capacity for the whole network.

required using this hierarchical addressing. Its relation with the underlying topology and possible reductions are considered later.

The minimum size for an address block corresponds to the set of hosts connected to the same router because smaller partitions do not provide any reduction in the number of rules. The identification of hosts and terminals is done similarly in TRILL and FabricPath. This addressing model is denoted as *per-switch* while the traditional non-hierarchical, flat addressing model is denoted as *per-host*. Larger groups can be formed by addressing several switches with a common group prefix, which is denoted as *per-group*, and forwarding traffic according to such group prefix. This three-level hierarchy, which splits an address in *host*, *switch* and *group*, reduces the number of flow rules because all hosts of a switch can be identified by solely one rule when the host information is not relevant. Similarly, all switches and hosts of a group can be identified by one rule when the particular switch and host information is not relevant. Figure 4-2 depicts these addressing models; the specific number of bits of each field may vary according to the controller policy.

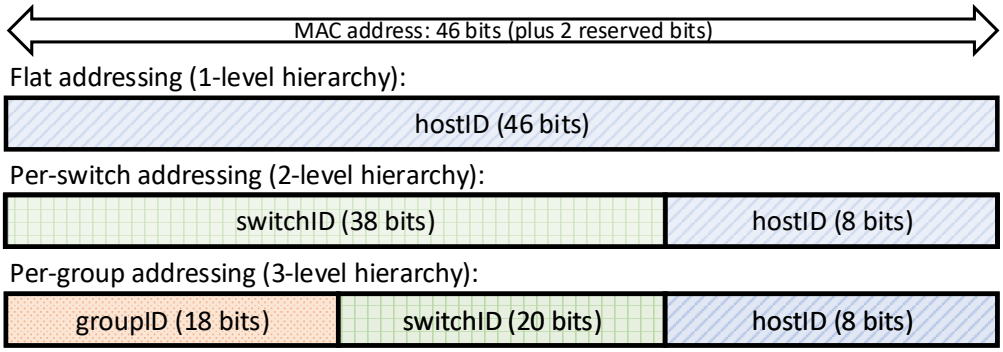


Figure 4-2: Traditional flat and two hierarchical addressing models.

The number of flow rules (R) required in each access switch with flat addressing model is equal to the number of hosts. Using the following notation: G is the number of groups if per-group addressing model is employed and $G = 1$ for a two-level hierarchy; S is the number of access switches per group considering one group if per-switch addressing model is employed; and I is the number of hosts on each access router; the number of flow rules is:

$$R = (G - 1) + (S - 1) + I. \tag{4-1}$$

The network size will be determined by the topology and switch size. The proportion of router ports used to connect hosts depends on the topology. Table 4-2 represents this value for several topologies: 3-level folded-Clos, *Flattened Butterflies* (FBs, [110]) and *Dragonflies* (DFs, [108]). Folded-Clos are *indirect* topologies with transit switches to which no host directly connects, so the overall proportion of host ports in the network is lower than in access switches. Flattened Butterflies and Dragonflies are recent *direct* topologies which are used in commercial HPC systems. Since direct topologies do not have *transit* switches, the

proportion of host ports in the network routers is the same that in access switches. The values of the table roughly determine the number I of hosts on each switch, for a given router-radix k. Following the notation introduced previously, the number of hosts (H) is:

$$H = G \times S \times I. \tag{4-2}$$

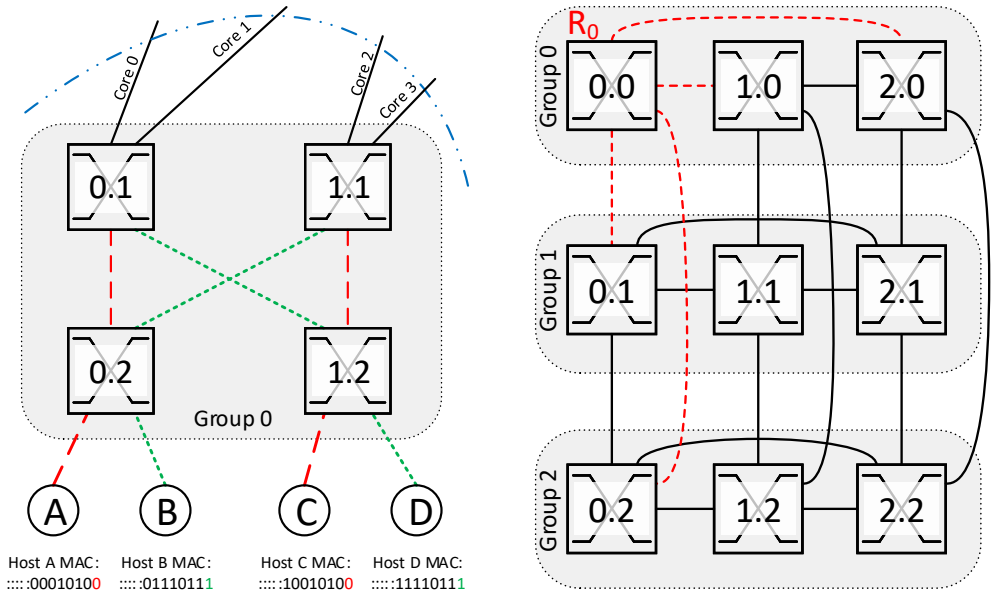
Table 4-2: Number of ports dedicated to compute hosts and network scalability in different topologies, using *k*-radix routers.

Topology	Hosts per access switch, I	Hosts per network switch	Scalability (max hosts, H)
3-Level folded-Clos	$\approx k/2$	$\approx k/5$	$k^3/4$
2-D Flattened Butterfly	$\approx k/3$	$\approx k/3$	$\approx (k/3)^3$
3-D Flattened Butterfly	$\approx k/4$	$\approx k/4$	$\approx (k/4)^4$
Dragonfly	$\approx k/4$	$\approx k/4$	$\approx 4 \times (k/4)^4$
4-D Flattened Butterfly	$\approx k/5$	$\approx k/5$	$\approx (k/5)^5$

The number of flow entries R required in access switches is minimized for a given network size if per-group addressing model is used and $G \approx S$. However, the number of switches assigned to each group might depend on the network topology; Figure 4-3 represents the three topologies considered in this chapter. In the three-level folded-Clos in Figure 4-3(a), nodes are organized in several *pods* which are connected via core switches with twice as many pods as access switches per pod. Its natural division is one group per pod. The 2-D Flattened Butterfly in Figure 4-3(b) has as many rows as columns, with an all-to-all connection per row and column. Matching a row to a group leaves $S = G$. With more dimensions, the same assignment of one group per row leaves $G > S$. Finally, the Dragonfly topology in Figure 4-3(c) is naturally organized in groups, but their amount is significantly larger than the number of switches per group.

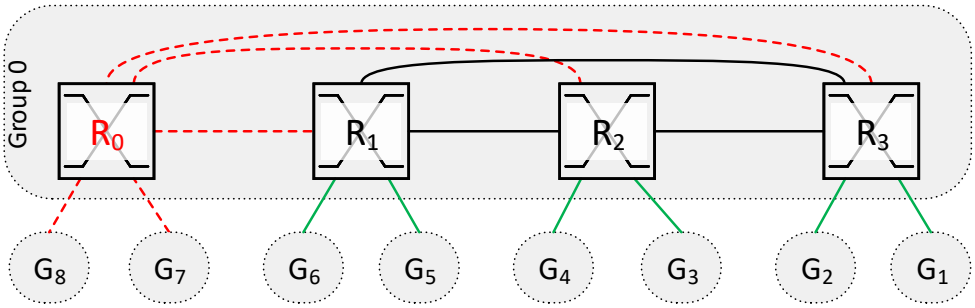
Considering this organization, Figure 4-4 depicts the number of flow rules required to support a given topology and size, considering flat, 2-level and 3-level hierarchical address organizations. *Flat addressing* represents the traditional CAM-based implementation with one entry per host and it is included for comparison. The scalability, which represents the number of hosts, only depends on the switch size and topology; the number of flow rules, in vertical axis, also depends on the use of *per-switch* or *per-group* addressing models.

The size of TCAM tables in commodity Ethernet switches usually ranges from 4K to 64K entries [114]. As observed in the plot, using *per-group* hierarchical addressing and DF or 3-D FB, it is possible to reach more than the 300,000 estimated hosts for future exascale HPC systems using a layer-2 network fabric requiring less than 4K rules.



(a) 3-level folded-Clos with k=4 ports routers. Only one *pod* is shown, the third upper level and the rest of pods are omitted for simplicity.

(b) 2-D Flattened Butterfly highlighting one router and their links. Each row corresponds to a group.



(c) Dragonfly with 9 groups, G_0 - G_8. Only group G_0 is detailed.

Figure 4-3: Different topologies being considered: (a) folded-Clos, (b) 2-D FB and (c) DF.

4.3.2 Scalability analysis with TCAM rules compaction

This subsection explores the use of wildcards for flow table compaction for the three previous topologies. Equation 4-1 considered one flow rule for each possible destination: host, switch or group. However, when several destinations share the same output link, they might be merged into a single rule depending on their addresses.

The minimum number of TCAM entries in a switch equals its port count, since at least one entry is required to output frames on each port. This lower bound is reached naturally in the 2-D Flattened Butterfly topology with per-group addressing. This occurs because each switch is directly linked to its own I terminals, the $(S-1)$ other switches in its own group (different columns) and the remaining $(G-1)$ groups (different rows) without overlap. This is highlighted in Figure 4-3(b) for switch R_0. For three or more dimensions, a similar reduction

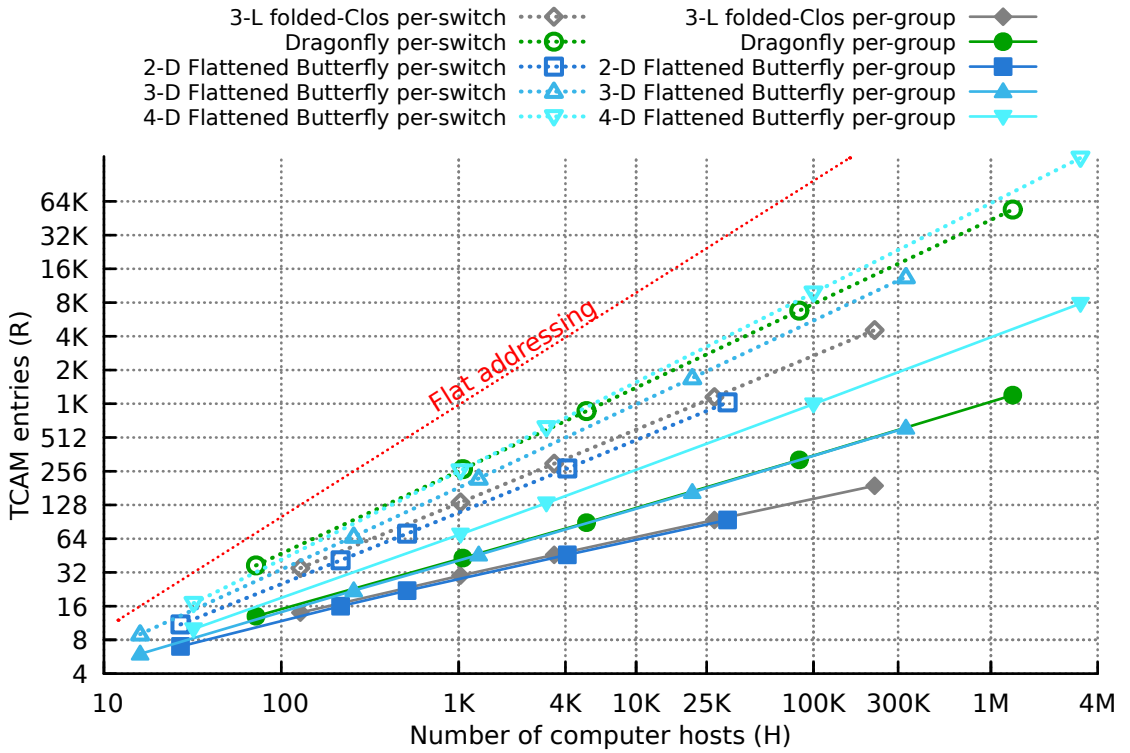


Figure 4-4: Number of TCAM entries required for varying network size and topology, using per-switch or per-group addressing. The points correspond to the maximum network size using switches with $k \in \{8, 16, 24, 48, 96\}$ ports.

is feasible, by simply assigning the switch location coordinates (X, Y, Z) to different sub-fields in the header as presented in Figure 4-5 and using wildcards in the corresponding sub-fields. Thus, a Flattened Butterfly of any dimension can reach the minimum number of rules.

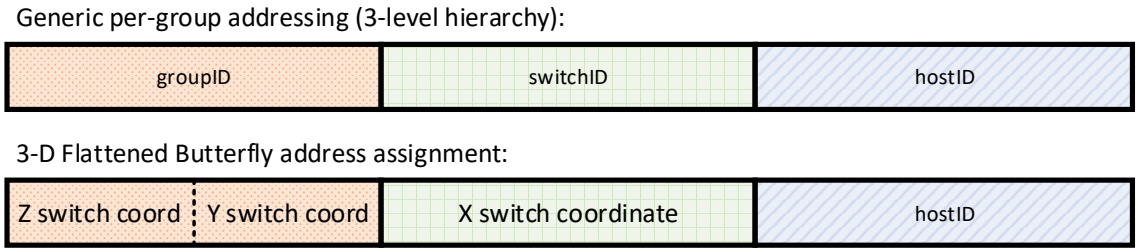


Figure 4-5: Hierarchical addressing in 3-D FBs considering forwarding rules compaction.

Figure 4-3(a) presents an excerpt of a folded-Clos, showing one pod and the uplinks to core routers (level three of the network). Routing in the folded-Clos is *up-down*. The uplink followed to core routers is independent of the destination, since all core routers are ancestors of all pods (the same applies to neighbor switches in the pod). Thus, a hash of the source fields can be used for static balancing of the uplinks, as in the example presented in the figure with different colors. Again this leads to a number of rules equal to the switch ports.

On the contrary, in the Dragonfly topology it is not possible to reduce the number of rules to the number of switch ports. To illustrate it, consider the leftmost switch in Group 0 in Figure 4-3(c). It requires one rule for each of its I hosts, one rule for each of the $S-1$ direct neighbors in the group, and one rule for each directly connected remote group, which equals the port count. However, it also requires additional rules to reach the groups connected to other switches in its group. At least, this would imply $S-1$ additional rules if all the rules associated to a given output port could be compacted. However, in the general case this is not possible because the addresses of the remote groups are not necessarily aligned, as in the example in the cited figure: using wildcards to compact rules for remote groups 1 and 2 would also match group 3.

Figure 4-6 shows the number of rules for the topologies considered. In the case of the DF, the maximum and minimum number of rules is presented; the actual number will be in-between and will vary from switch to switch. In the other cases, the lower limit is reached. Even in the worst case of the DF with more than 1 million nodes, the number of rules is relatively low and fits in existent OpenFlow switches.

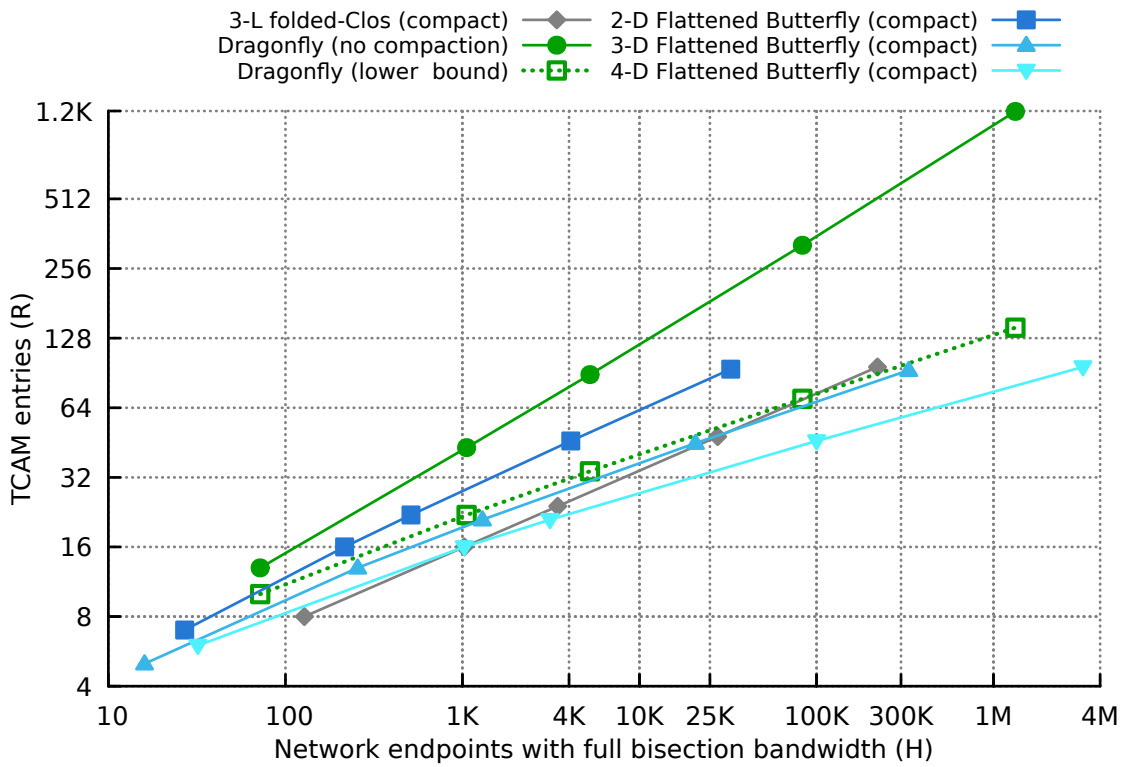


Figure 4-6: Number of TCAM entries after compaction. The points correspond to the maximum network size using switches with $k \in \{8, 16, 24, 48, 96\}$ ports.

4.4 MAC address rewriting

This section identifies a problem generated by *dynamic* or *on-the-fly* rewriting of MAC addresses and introduces an alternative mechanism to automate host MAC configuration. The scalability solution in Section 4.3 requires rewriting MAC addresses to include location information. Previous proposals implement dynamic rewriting of the frame headers in access switches [143, 163]. However, this interferes with layer-2 service announcement mechanisms, used for example in Open-MX.

Distributed discovery mechanisms rely on requests from the service users, such as ARP for IP-to-MAC mapping, or announcements from the service provider, such as Open-MX computing nodes announcing their availability. In the first case, requests can be intercepted by the access switches and derived to a central fabric manager, as done in PortLand [143] and SEATTLE [106] to attend ARP requests. In the second case, by contrast, broadcast and multicast messages announce the service to all nodes, so a similar centralized solution is not feasible. Furthermore, MAC rewriting interferes with these announcements. In fact, Open-MX announcements include the source node MAC address in the data field, so receivers of this frame record the original unmodified address and communication never happens.

Two alternative solutions are introduced next. The first one modifies the service discovery protocol, with receivers recording the MAC address in the announcement frame header, rather than the data field. This is compatible with on-the-fly address rewriting and is possible in Open-MX because it is open-source, but it might be unfeasible with proprietary stacks.

The second alternative modifies the MAC address in the hosts configuration, rather than modifying it in the frame headers. Since individually modifying each host's address is unfeasible with hundreds of thousands of nodes, a control protocol has been designed to automatically configure the host MAC at boot time. This protocol is denoted as *Dynamic MAC Protocol* (DMP) and it is depicted in Figure 4-7. When a host boots, it asks for a MAC address by sending a DMP request using a specific EtherType value. Its own switch identifies this query and sends the frame back, overwriting the source address field with the new hierarchical MAC address, as discussed in Section 4.3. One rule per connected host is required for DMP, in addition to the routing rules discussed in the previous section. This approach does not restrict the use of the host MAC address in the announcement payload nor the use of hierarchical routing.

Both alternatives have been implemented and verified in a computer using *Open vSwitch* and four virtual machines. In these tests OpenFlow 1.0.0 [145] has been used with the optional action *modify-field* to change the header MAC field. The *EtherType* value 0x88b5 reserved for experimental purposes [91] has been used on DMP packets in the experiments.

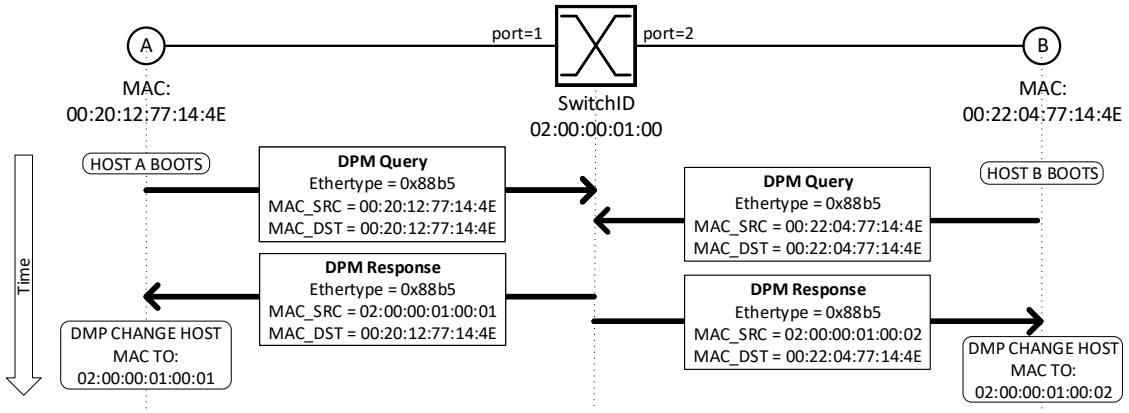


Figure 4-7: Sequence of packets when two hosts boot using the DMP protocol.

4.5 MAR-bP: Multipath Adaptive Routing based on Pauses

As a first approach to the objective of this chapter, this section aims to design a non-minimal adaptive routing over a hardware as much close as possible to commodity Ethernet network device with OpenFlow support. To do that, it explores adaptive routing in Ethernet networks based on Ethernet *pause* messages [88] resulting in *MAR-bP: Multipath Adaptive Routing based on Pauses* proposal. Oblivious multipath routing employs a fixed function for load balancing, typically a hash of some header fields. Such approach is used in traditional *link aggregation* [86] or *equal-cost multipath* [83] routing algorithm. Its disadvantage is that the paths might receive a different load, leading to suboptimal throughput. By contrast, adaptive routing dynamically balances the load of each path. An adaptive routing implementation is more complex since it requires an estimation of the network load and instantiating additional routing entries.

Section 4.5.1 introduces *conditional flow rules* triggered by pause flow-control messages. This is applied to both minimal routing, in Section 4.5.2, and non-minimal routing, in Section 4.5.3. A discussion is presented in Section 4.5.4.

4.5.1 Proactive conditional flow rules

Section 4.2 concludes that reactive flow rules instantiated by the controller are not well suited to adapt switch routing to changing HPC traffic. To solve the aforementioned problem, the idea of *conditional flow rules* is proposed. These rules are proactively instantiated by the OpenFlow controller, and they are deactivated locally by the switch on network events, such as link-level flow-control pauses. Using this idea, traffic is diverted to alternative paths when a preferred one gets blocked due to congestion.

Figure 4-8 depicts the idea of conditional flow rules with the required changes to the flow table. Besides *ordinary* rules, which indicate the output port associated with each terminal connected to the switch, the system relies on two sets of rules: *default* and *alternative*,

with different priority. Both ordinary and default rules correspond to hierarchical addressing model, explained in Section 4.3, and they have high priority, whereas alternative rules employ low priority. Default rules are conditional, and therefore dependent on the *pause status* of their associated output port which is highlighted in Figure 4-8. Commonly, both default rules and their corresponding alternative backup rules match for a given frame, but default rules are used because of their higher priority. However, when a given output port receives a *pause*, the corresponding default rules are deactivated, so alternative rules start to be used, diverting traffic.

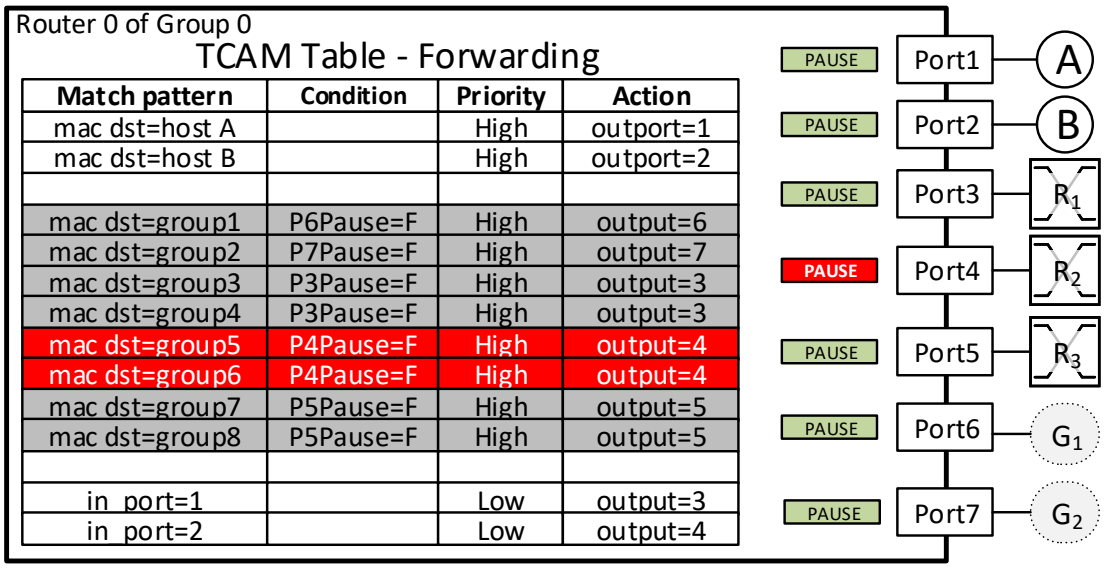


Figure 4-8: Router's architecture with conditional flow rules. When the condition fails, i.e., when the output port is paused, the high-priority minimal routing rule is ignored, leading to the use of a low-priority rule.

With this proposal, default paths are used until their queues become completely full. Both default and alternative rules are proactively instantiated by the controller after topology discovery. Their application for minimal or non-minimal multipath routing is discussed in the next sections.

4.5.2 Conditional flow rules for minimal routing

Conditional flow rules can be applied to topologies with multiple minimal paths by assigning a *default* rule to each of the default paths, and one or more *alternative* rules (for each alternative minimal path using different levels of low priority) to paths already assigned to other default rules.

In the case of the folded-Clos with compact routing rules, one default rule is assigned to each uplink using per-source hashing, although per-destination hashing is equally possible. Up to $k-1$ additional alternative rules can be assigned to the same per-source hash, for the $k-1$ remaining uplinks. This multiplies the number of uplink rules in a factor up to k.

In D-dimensional Flattened Butterflies there are up to D potential outputs for a given destination, for the D options of the first hop. Interestingly, in this case multipath routing can be implemented without increasing the number of rules with respect to the compact case. Higher-level rules (e.g., to reach a remote group) will be conditional with high priority, while lower-level rules (e.g., destination switch index) will have low priority and employ wildcards on the group bits.

Since there is no minimal path diversity in a Dragonfly topology, using conditional flow rules to enable multiple minimal routes does not apply. However, each access switch requires I ordinary rules to eject the packets to the I terminals connected to it and one rule per remote group to route the packets to them minimally.

4.5.3 Conditional flow rules for non-minimal routing

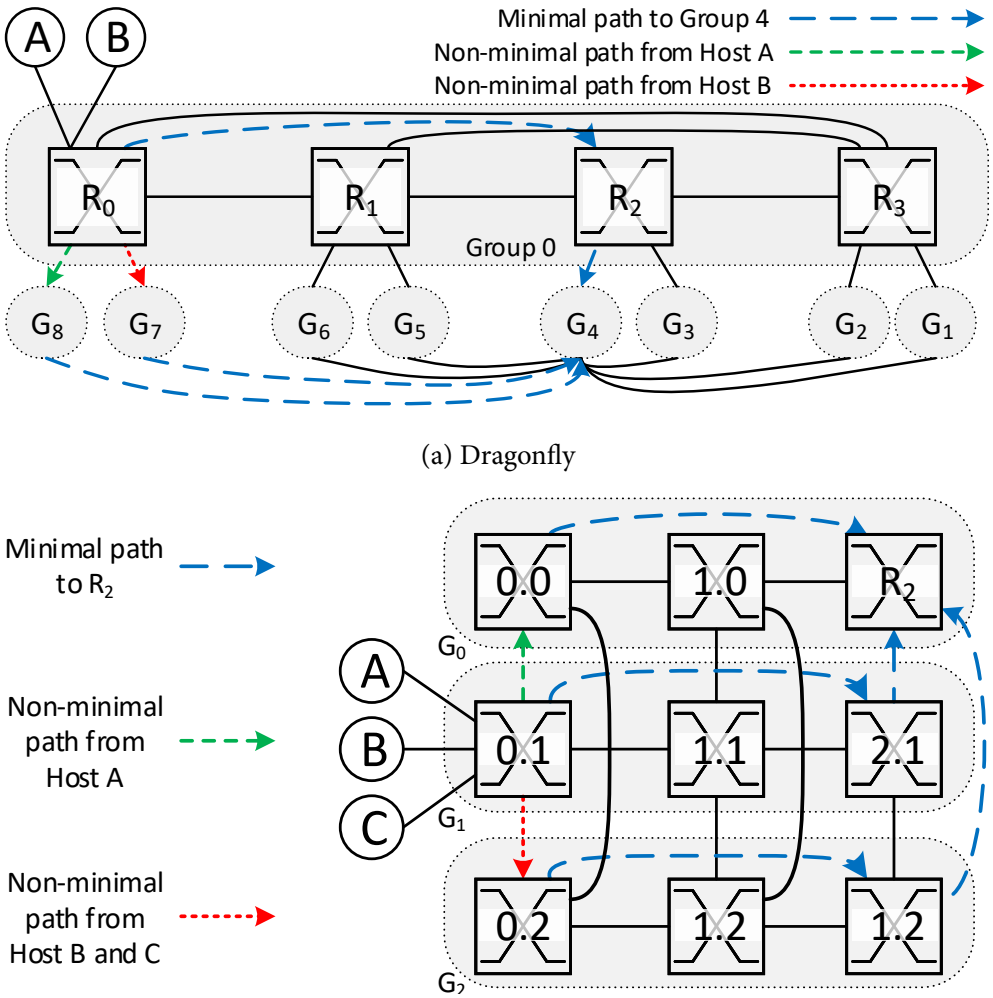
Non-minimal routing using conditional flow rules relies on diverting traffic to a given intermediate destination when the corresponding minimal path is congested. The intermediate destination cannot be selected randomly without intervention from the controller, so a given Valiant intermediate router will be statically assigned to each source node. The application to Dragonflies and Flattened Butterflies is depicted in Figure 4-9.

Figure 4-9(a) shows the application of conditional *non-minimal* routing to Dragonfly network. In this topology, global links are the most congestion-prone ones. Each switch employs several high-priority rules for minimal routing as discussed in Section 4.3.2. Of these, rules for remote groups will be conditional. I additional low-priority rules, one per each terminal, are included to forward traffic from injection ports to remote groups. Note that the input port can be checked in OpenFlow rules. When the conditional minimal rule is deactivated, these additional rules forward traffic directly to global links, towards a remote intermediate group. From that point, minimal routing is employed. Since a balanced Dragonfly has as many terminal per switch ($I = p = h$) as global ports [108], one non-minimal output can be assigned to each compute hosts, effectively balancing traffic. This mechanism to enable non-minimal routing requires I additional rules and is equivalent to VLB_{-g-}.

The application to Flattened Butterflies is similar, as depicted in Figure 4-9(b). In this case, each switch is connected to $S-1$ remote groups but there are $I = S$ compute hosts per switch to achieve a balanced design [110], so one of the hosts will not have a non-minimal path assigned (or one of them will be repeated).

4.5.4 Discussion

Deadlock issues, explained in Section 2.5, must be managed. For folded-Clos, simple up-down routing avoids deadlock. As explained in that section, the proposal used in this chapter relies on the use of multiple *Virtual Channels* (VCs). In particular, D VCs are required for fully adaptive routing in D–dimensional Flattened Butterfly and just three VCs are required



(a) Dragonfly

(b) Flattened Butterfly

Figure 4-9: Non-minimal routing in (a) DF and (b) FB networks using conditional flow rules.

for Dragonfly, which can be reduced to two in the proposed *MAR-bP* adaptive routing implementation since there are at most two hops of each type. These VCs can be mapped directly to two different Ethernet *Class-of-Service* (CoS) levels and different switch buffers, leaving enough CoS levels to differentiate other types of traffic. Class-of-service updates can be embedded in existent OpenFlow rules.

The proposed adaptive routing decision relies on snooping pause link-level flow-control messages. Alternative implementations might rely on explicit congestion notifications, as proposed in the next section.

Adaptive routing mechanisms can increase traffic throughput at the cost of out-of-order delivery. It is the responsibility of the transport protocol to detect and reorder network traffic in such cases, these protocols are out of the scope of this chapter.

The use of a statically pre-selected Valiant path differs from the original random definition. Additional non-minimal paths can be included using different levels of low priority, at the cost of an increased number of rules.

4.6 QCN-Switch: adaptive routing based on ECN messages

This section introduces *QCN-Switch*: a statistical *non-minimal source-adaptive congestion-aware* routing algorithm relying on *explicit congestion notification* messages which extends the *MAR-bP* routing algorithm introduced in the previous section.

Many previous mechanisms implement table-based adaptive routing that performs load balancing. In previous proposals, balancing is typically performed with the granularity of a packet [108, 96, 74], flowlet [99] or flow [62, 137, 63, 159]. However, those proposals either employ a congestion-oblivious balancing function, which implies that they do not really adapt to changing network congestion, and do not support non-minimal adaptive routing or rely on packet-by-packet network information such as credit counters, with minimal feedback delay.

The proposed design adapts routing to network traffic by capitalizing on *Quantized Congestion Notification* (QCN), which is detailed in Section 2.4.1, congestion messages generated when transit buffers are heavily used. *Congestion Notification Messages* (CNMs) update network status with a coarse granularity, in the order of hundreds of packets. This is affordable for injection throttling, because injection rate is modulated in an almost-continuous range, from zero to the maximum interface rate. By contrast, table-based adaptive routing needs to select the destination between a small set of forwarding entries, so large notification delays would fix the selection of a given path and overload some network area. Indeed, a large feedback delay combined with a deterministic selection function in the forwarding table may cause abrupt traffic oscillations.

This proposal extends the *conditional flow rules*, previously introduced in Section 4.5.1, to forward traffic minimally or non-minimally based on a probability, as it is explained in Section 4.6.1. The proposed *QCN-Switch* does not modify the QCN *Congestion Point* (CP) whereas the QCN *reaction point* in NICs is disabled. Instead, source switches, which are directly connected to the destination NIC of a CNM, intercept QCN notifications and process them in order to modify the aforementioned probability according to two different policies. A *base* probability management variant, explained in Section 4.6.2, modulates the probability of each output according to an *Additive-Increase, Multiplicative-Decrease* (AIMD) policy. This base design is extended in Section 4.6.3 with a *feedback comparison* probability management variant, which targets the issue of throughput fall at uniform high loads present in the base. Finally, a *source processing* mechanism that enable switches to process their own CNMs is presented in Section 4.6.4.

4.6.1 Forwarding tables with probabilities

To provide a smooth balance in the utilization of different network paths between the reception of consecutive congestion notifications, this proposal relies on conditional flow rules

with *statistical* path selection. Path selection is modulated with a set of control values derived from the congestion status reported by the network. Such control values are calculated to set the *probability* of forwarding traffic minimally using a given output port, so one value is required per transit port. These values are denoted as *minimal port probability* or *port probability*, and their range is 0-100%. The previous design of conditional flow rules is similar to the introduced in this section, but it uses a binary probability value that is 0% or 100%.

To implement the statistical path selection, routing table entries with priorities are considered. Priorities are used to discern minimal and non-minimal forwarding table entries: high-priority conditional rules are used for minimal routing and low priority rules are used for non-minimal routing. Routing is calculated so that, when path selection is possible, one high and at least one low priority entries match a given packet. Source-adaptive routing is considered during the evaluation, so path selection only occurs at injection and non-minimal rules are applied only for injection ports, but the model might be applied to in-transit adaptive routing. When only one rule match occurs, such rule is applied regardless of its probability value.

A random number N, which ranges from zero to hundred, is employed when multiple rules with different priorities match a packet, as follows. If N exceeds the probability of the output port of a matching high-priority conditional rule, this rule is not executed. Instead, the matching lower-priority rule is followed. The random value N may be generated from the incoming frame, by selecting some bits from its CRC for example, or derived from a pseudo-random sequence.

Figure 4-10 shows an example of the proposed routing table and associated port probabilities for one switch in a Dragonfly group. The switch has five transit ports: ports 3-5 connect to other switches in the same Dragonfly group, and ports 6-7 connect to remote groups. Table entries for minimal routing have *high* priority and employ a condition that is associated to the probability of their output port; table entries for non-minimal routing have *low* priority and are not conditional. In this example, port 5 has received large congestion notifications and has reduced its probability to a low value (10%).

The random value generated for the query shown in Figure 4-10 is $N = 77$. Thus, probability values lower than N disables minimal routing paths via ports 5 and 7 which are known to be congested. When the condition is false, an alternative non-minimal path is selected based on the input port of the packets.[10] Regarding the example in the figure, the associated table entries disabled are highlighted in red and alternative non-minimal paths are determined by the two entries in the lower part of the table. If there is not an alternative entry, the condition will be ignored; this avoids non-minimal forwarding of in-transit traffic and creating forwarding loops.

The probability of each conditional rule needs to be adapted to the congestion level in the corresponding path, which is estimated from the feedback values received in the QCN con-

[10]Note that in certain cases the alternative rule might overlap the minimal path.

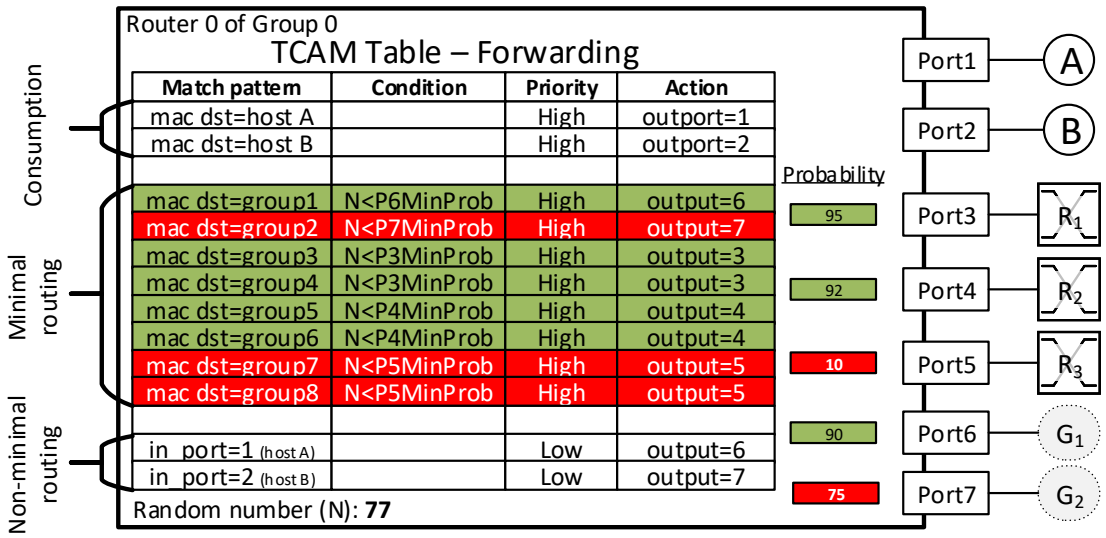


Figure 4-10: Router architecture with QCN-Switch proposal. This diagram portrays R_0 of G_0 under adversarial traffic pattern for a (*h=2, p=2, a=4*) Dragonfly network. The condition or a rule fails when N is higher than the probability of sending minimally by its associated port. In such case, high-priority minimal routing rules are ignored (highlighted in red) leading to the use of a low-priority rule.

gestion notification messages. Different policies may be considered to update the probability values of each port; the base mechanism and one extension are presented next.

4.6.2 Base AIMD probability management

The *base* mechanism employs an AIMD policy to reduce the probability associated to an output port when a CNM is received through it, modulated by the *Feedback value Fb* indicated in the explicit congestion notification message. Since QCN only considers negative congestion notification messages and it does not notify the absence of congestion, the proposed policy protocol is implemented as follows.

- Upon reception of a CNM with feedback *Fb* through a port, the probability of sending the traffic minimally through this port is multiplied by a factor *R*, calculated as:

$$R = 1 - |L_f \times Fb|, \tag{4-3}$$

  where L_f is a limiting factor that determines the extent to which the probability decreases and *Fb* values range from 0 to 63 as presented in Section 2.4.1. For example, with $L_f = \frac{1}{128}$, *R* will be in the range between 0.5 and 1.

- A counter tracks the packets transmitted by each transit port between the reception of CNM messages. The counter is reset every time a CNM message is received and it is incremented for every transmitted packet. If it reaches a threshold PC_I, the probability value associated with that port is increased by *PI*%.

Parameters L_f and PI regulate the reaction time of this mechanism to changes in congestion: L_f determines how fast traffic is sent non-minimally after congestion is detected; analogously, PI indicates how quickly routers revert to minimal routing after congestion disappears. The impact of these parameters is evaluated in Sections 4.7.4.3.2 and 4.7.4.3.3.

4.6.3 Feedback comparison probability management

Non-minimal adaptive routing needs to react to network traffic conditions to avoid congestion under adversarial traffics. However, the information received from a single individual port is not enough to accurately identify an adversarial traffic pattern. It should be noted that an adversarial traffic pattern will cause congestion in one or a low number of ports while a heavily loaded uniform load will cause congestion in all the ports.

When the network is heavily loaded under a uniform traffic pattern, all network buffers get similarly loaded and all switches generate some congestion notifications. In this case, switching to non-minimal routing because notifications are received from an individual port is counterproductive: non-minimal routing increases the load in the network, which eventually generates more congestion notifications and further non-minimal routing, in a positive control loop. The effect is that all traffic tends to be forwarded non-minimally, with reduced throughput and increased latency.

Feedback comparison calculates an average feedback value Fb_{avg} which represents the average congestion of all transit ports of a switch. This value is calculated from the most recent feedback values $Last\ Fb_i$ received on each of its transit ports (i) according to the following expression:

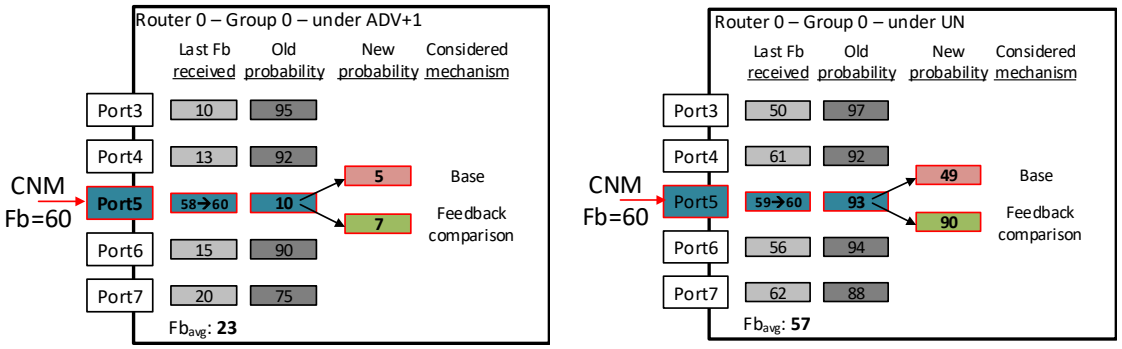
$$Fb_{avg} = \frac{\sum_{i=transit\ ports} |Last\ Fb_i|}{i}. \tag{4-4}$$

Upon reception of each QCN message, the switch recalculates Fb_{avg} and compares this average with the feedback received in the CNM. If $|Fb| < Fb_{avg}$, the probability associated to that port is increased by PI%, same as occurs when no CNM is received for an interval in the base policy. If $|Fb|$ is greater, the port's probability is reduced by the factor:

$$R = 1 - L_f \times (Fb - Fb_{avg}). \tag{4-5}$$

Figure 4-11 shows an example of probabilities update using both probability management variants: *base* and *feedback comparison*, under uniform or adversarial traffic. The use of Fb_{avg} to calculate the reduction factor R in the latter reduces the impact of congestion notifications on uniform loads, when all ports experience similarly high congestion values.

Buffer occupancy naturally suffers ephemeral variations, which is denoted as *transient congestion* in the literature [192]. Therefore, congestion notification feedback values may change significantly even in a stationary state, which may increase variability. The feedback comparison probability management variant employs two thresholds Th_1 and Th_2



(a) Adversarial traffic pattern.

(b) Uniform traffic pattern with high load.

Figure 4-11: Example of probability values update when a CNM with *Fb=60* arrives, under two different traffic scenarios for a switch in a Dragonfly network (*h=2, p=2, a=4*). Under uniform traffic the port probability remains high when using the *feedback comparison* variant. Changing values are highlighted in red.

to eliminate minor changes due to such transient variations. Port probabilities are not modified when the feedback value received in a notification lays within the range ($Fb_{avg} - Th_1, Fb_{avg} + Th_2$), as depicted in Figure 4-12. The impact of these thresholds is evaluated in Section 4.7.4.3.4.

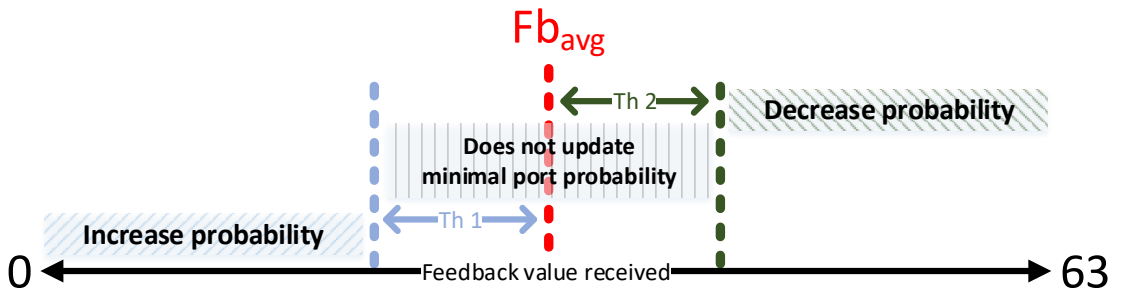


Figure 4-12: Thresholds used in the feedback-comparison probability management variant.

When network congestion disappears, CNMs might be no longer received given the lack of positive, i.e., lack of congestion, notifications in QCN. In this case, Fb_{avg} might remain with a high value with the previous implementation. While this potential issue has not been observed during the simulations,[11] a realistic implementation would need to adapt Fb_{avg}, for example reducing each *Last* Fb_i according to a negative exponential decrease when no CNMs are received.

4.6.4 Source processing mechanism for input sensing

Depending on the switch internal speed-up, congestion trees, also known as high-order head-of-line blocking [97], can be rooted at input or output ports [75], but with sufficient

[11] A case of congestion disappearing is presented in Figures 4-19(c) and 4-19(d).

speed-up this typically occurs at the outputs. The QCN-Switch adaptive routing intercept CNMs messages, which are generated by QCN CP according to Section 2.4.1, in order to adapt routing. As explained, the congestion sensing can be done at input or output buffers. However, a pathological case of unfairness can occur specifically when congestion sensing is implemented at the input buffers. Consider the situation of adversarial traffic presented in Figure 3-5 on page 54. Congestion occurs in the minimal path, saturating input buffers in local ports and the output buffer in the global port of R_{OUT}.

With output-buffer sampling, R_{OUT} generates CNMs indicating congestion associated to the output buffer of its global port; these CNMs are sent to all the eight nodes in the example, and they are intercepted by their associated access router (R_0, R_1, R_2 and R_{OUT} itself). In the case of R_{OUT}, this CNM reduces the probability of the output port that generates the CNM, whereas in the neighbor switches the probability that is reduced is associated to the port that receives the CNM.

By contrast, with input-buffer sampling, CNMs are generated based on the occupancy of the three buffers associated to local ports connected to the other source switches in the example, and they are sent only to the neighbor switches. No CNM targets the two computing nodes associated to R_{OUT}, because their input buffers are not associated to transit ports. Additionally, if the message was generated, it would need to be processed locally without information about which is the congested output port. Therefore, R_{OUT} never modifies its own output port probabilities and always sends traffic minimally, suffering much higher congestion. For this reason, nodes in R_{OUT} may inject a much lower amount of traffic.

The *source processing* mechanism attacks this limitation trying to resemble the advantage of sensing at output buffers which allows that generated CNMs can be sent to all sources in the network regardless of their source switch. The introduced mechanisms enables all switches to process their own CNMs generated based on congestion at input ports. When a CNM is generated by a QCN congestion point in a switch, it is sent to the source as usual but it is also processed in that same switch, as if it had been received from a neighbor switch. In this case, the feedback of this QCN message is used to decrease the probability of the output port used to forward the *victim packet* selected in the sampling process. This allows all switches in a group to detect congestion and adapt their tables in response.

The evaluation of both sampling alternatives are presented in Section 4.7.4.1.1, including a complete implementation which combines *QCN-Switch* with *source processing* and *feedback comparison*.

4.7 Evaluation

The particular methodology employed in this chapter is presented in Section 4.7.1. Section 4.7.2 presents an evaluation of the network power consumption for different topologies and TCAM organizations proposed in Section 4.3. Then, the performance results of the

proposed adaptive routing in Sections 4.5 and 4.6 are detailed in Sections 4.7.3 and 4.7.4 respectively.

4.7.1 Methodology

This section aims to introduce the aspects of the methodology which are particular to this chapter and are not presented in Chapter 3. This includes the methodology used to determine the power consumption and the particular simulator configuration employed.

4.7.1.1 Power consumption

The power evaluation performed in this chapter takes into account the topology to be able to calculate the consumption of electrical and optical SerDes. It also considers the power consumption of the network devices' CPU, buffers and, CAM or TCAM.

Models of power consumption of binary CAMs [84] or ternary CAMs [6] tables show that power scales roughly proportionally with table size. In particular, the main power consumption in TCAMs comes from its *match-line* logic, which grows linearly with the number of entries. The consumption of these tables typically reaches several tens of watts [6, 101], being the most hungry modules besides ports SerDes [48]. The tool available in [7] are used to calculate the power consumption of TCAM tables considering 32 nm CMOS technology and 1,000 header bits, which can be matched by OpenFlow 1.5.

4.7.1.2 Simulator configuration

The simulation experiments designed to evaluate the performance of non-minimal adaptive routings proposed have been carried out according to the methodology explained in Chapter 3. Table 4-3 lists the simulation parameters concerning the proposals of this chapter and any modification over the base simulation parameters exposed in Table 3-1. The network simulator mimics the behavior of conditional flow rules as described in aforementioned sections. Packet size is set to 1 KB as an intermediate value between minimum and maximum packet size for Ethernet technology. Network links have a bandwidth of 40 Gbps and latencies of 40 ns and 400 ns respectively for electrical and optical ones, which correspond to cables of 8 and 80 meters. Four Ethernet CoS levels, using per-priority flow control for a lossless implementation, are considered for deadlock avoidance and to prioritize QCN congestion notification messages over the rest of traffic.

QCN congestion points have been implemented following the standard [10], including the feedback calculation and timing parameters indicated in Section 2.4.1. The recommendation to set Q_{eq} to 20% of the size of physical buffer is followed and it is also maintained the selection of 1 packet from each 100 for the sampling in the CP (denoted as *Congestion Point Cycle*, CPC); this parameter is also part of the sensitivity analysis performed in the evaluation. Another parameter for this study is the percentage of samples with a negative

Table 4-3: Particular simulation parameters used in this chapter.

	Parameter	Value
Network configuration	Link speed	40 Gbps
	Packet size	1,000 bytes
	Switch latency	200 ns
	Local link latency	40 ns (8 m)
	Global link latency	400 ns (80 m)
	Class-of-service levels	4
	Injection queues size	200 KBytes
	Transit queues size	100 KBytes
QCN	Queue's reference point	Q_{eq} = 20% of queue size
	Weight value	$w = 0$
	Congestion point cycle	CPC = 100 packets
	% of samples which actually generate a CNM	$\%CNMs$ = 30%
QCN-Switch	Reduction limiting factor	$L_f = 1 / 64$
	Packet counter limit	PC_l = 100 frames
	Probability increase	PI = 1 %
	Feedback comparison thresholds	$Th_1 = 0$, $Th_2 = 30$

feedback value that actually generates a CNM; a default value of 30% is considered, and this selection is justified in Section 4.7.4.3.1. QCN CPs are implemented at the input ports of the switches, as suggested in [142], or at the output ports [10] to be able to compare both alternatives. Preliminary explorations in [24] found that a parameter $w = 0$, which omits the velocity in Equation 2-3, led to better results than the default $w = 2$. QCN reaction points, which implement injection throttling in the NICs, are disabled to focus the evaluation on in-network congestion.

Any change of the default values shown in Table 4-3 will be clearly stated during the evaluation. QCN standard configuration parameters and tuning possibilities of QCN-Switch are described previously. A reduction limiting factor $L_f = 1/64$ is employed, which allows for output port percentage reductions in a factor up to $R = 1 - 63/64 = 0.0156$. This guaranties a very fast transition from minimal to non-minimal routing. By contrast, a default probability increase of $PI = 1\%$ is employed, which corresponds to a slow return to minimal routing when congestion disappears. Such design is sensible, since minimal routing in presence of

adversarial traffic in Dragonflies is much more limiting than non-minimal routing under benign traffic. This is explored in Sections 4.7.4.3.2 and 4.7.4.3.3.

4.7.2 TCAM compaction and topology power comparison

This section compares the power consumption of the three topologies analyzed across the chapter: folded-Clos, Flattened Butterfly and Dragonfly. The network supports ≈ 300,000 hosts with full bisection bandwidth, built using 72-port Ethernet switches. Power calculations consider 40 Gbps ports and worst-case 64-byte packets. The energy required for reading and writing a 64-byte packet from the buffer memories is 4.5048 nJ and 4.4993 nJ respectively, following the calculations in a previous work [48]. Optical ports consume ≈ 0.6 W (4 lanes of 150 mW each) and the electrical ones, used to connect compute hosts in access switches, a 20% less [4]. A fixed value of 30 W has been considered for the CPU and logic.

The number of switches differs per topology. In folded-Clos, a 4-stage topology is required for the desired size. The design relies on basic *pods* with 36 switches in the first and second levels and 1,296 hosts. 1,296 stage-3 switches connect 36 pods in a stage-3 group, which is replicated 6.5 times to reach 29,484 switches and 303,264 hosts. The Flattened Butterfly requires four dimensions, with routers organized in a $15 \times 15 \times 15 \times 6$ array to reach 20,250 switches and 303,750 hosts. Finally, the Dragonfly employs 463 groups of 36 routers and 648 compute hosts, leading to 16,668 switches and 300,024 terminals. These networks do not reach the maximum size of each topology; in all cases, additional links in spare switch ports are considered to provide full bisection bandwidth.

Both CAM and the four TCAM organizations from Section 4.3 are compared: *CAM-flat* and *TCAM-Flat* represent a traditional Ethernet with per-host addressing, *Per-Switch* and *Per-Group* are 2-level and 3-level hierarchical addressing model, respectively and *Compact* is the TCAM compaction presented in Section 4.3.2. In the folded-Clos, the 3-level organization considers a *pod* to share a common group prefix. In Flattened Butterfly, a group is the set of routers in the first dimension whereas in the Dragonfly, each group is mapped directly to the group concept.

Figure 4-13 presents power results. In all topologies, the use of flat addressing with TCAMs is clearly unfeasible, since these huge tables would consume more than 900 W per switch and at least 15 MW overall.[12] Per-switch addressing reduces these values to be competitive with the original CAM approach, but the overall consumption ranges around a MW. Per-group addressing significantly reduces TCAM power, getting very close to the optimal solution based on compaction. Once TCAM power is minimized, the impact of topology (which determines the number of switches) is what drives power consumption. The Dragonfly topology, besides not being able to fully compact TCAM flow entries, offers the best result. Compact TCAM per-group Dragonfly setup reduces 54.1% of the original power in

[12] Resulting in an overall interconnection network power consumption of 19,3 MW, which is almost the projection for the maximum power consumption for an overall exascale system [111].

the reference CAM-flat folded-Clos from a saving of 17.0% due to TCAM compaction and 37.1% due to topology changes.

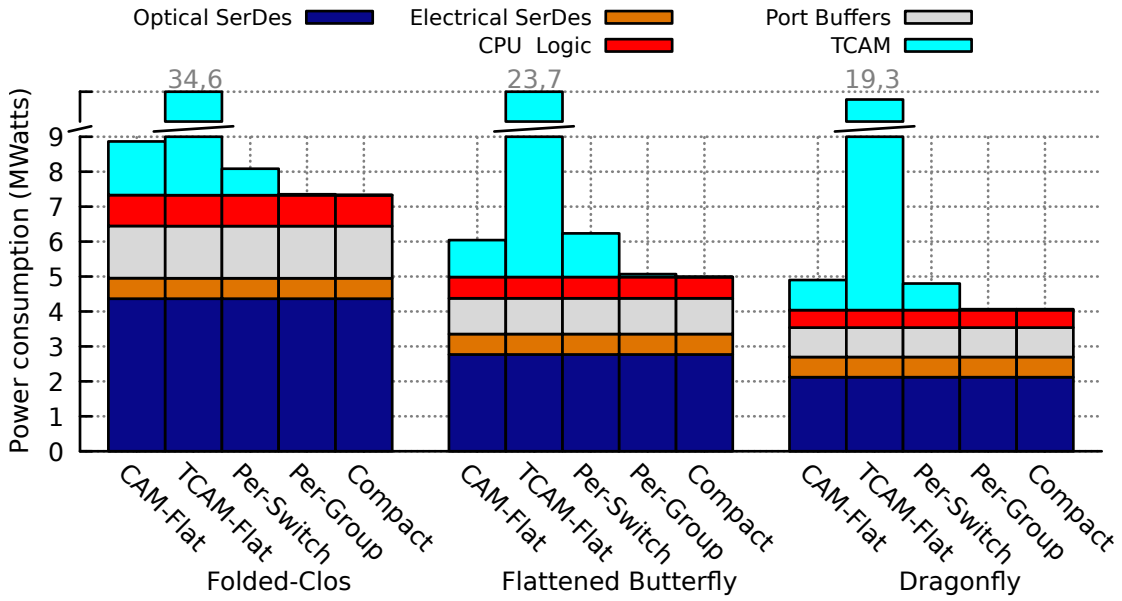


Figure 4-13: Network power consumption dissection.

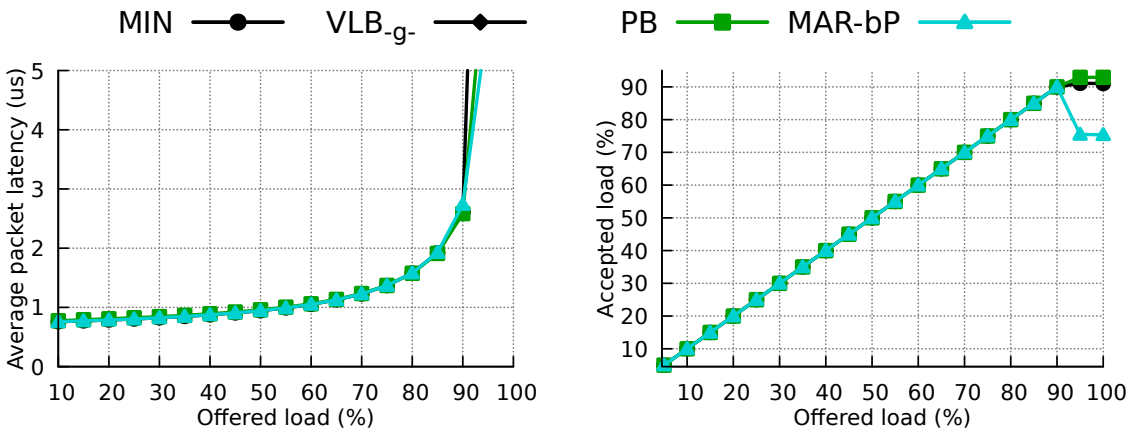
4.7.3 MAR-bP performance results

This section describes the performance results of MAR-bP: *Multipath Adaptive Routing based on Pauses*, presented in Section 4.5, including the design of conditional flow rules for minimal and non-minimal routing.

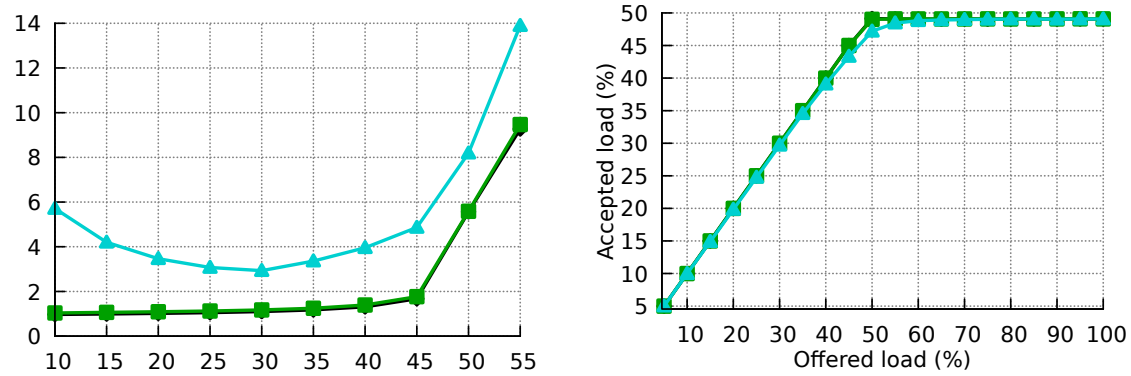
The interconnection network simulator mimics the behavior of conditional flow rules as described in Section 4.5.1. *Random Uniform* (UN) and *Adversarial shift* (ADV+*i*, with offset $i = 1$) traffic patterns have been considered. *Piggyback* (PB, [96]) routing algorithm is used as source-adaptive reference; it implements per-packet adaptive routing relying on state information for every global channel in a group distributed among switches within that group. Additionally, *Minimal* (MIN) and *Valiant load-balancing* with a maximum-length *phase A* path of one hop (VLB_{-g-}) routing algorithms are the oblivious references for UN and ADV+1 respectively.

Figure 4-14 shows average packet latency and throughput results under both above-mentioned traffic patterns. Under UN traffic, *MAR-bP* latency resembles the reference *MIN*, since traffic distribution does not generate congestion that do not trigger non-minimal routing. *PB* sends part of the traffic non-minimally which increases latency. In contrast, under ADV+1 traffic, MAR-bP average latency under small loads is larger than VLB_{-g-}. This occurs because conditional flow rules rely on flow-level *pauses*, which implies that minimal paths need to be saturated for non-minimal routing to occur. Hence, the amount of traffic

routed minimally experiences larger congestion and increases average latency. Still, the average packet latency of the proposed MAR-bP routing proposal against optimized custom HPC routing alternatives is competitive in medium and high loads.



(a) Average packet latency and throughput under uniform traffic.



(b) Average packet latency and throughput under adversarial traffic.

Figure 4-14: Average packet latency and throughput under (a) UN and (b) ADV+1 traffic patterns.

MAR-bP throughput under uniform traffic falls down due to congestion. However, the reference routing achieve a good throughput close to the theoretical maximum. The throughput result of MAR-bP under adversarial traffic is greater than the reference oblivious and adaptive routing references used because it uses non-minimal paths solely when minimal paths are saturated, so the minimal paths are used completely and after, the traffic follows non-minimal ones.

The issue detected in the latency results under adversarial traffic pattern and the throughput drop observed under uniform traffic pattern limit the viability of MAR-bP. Hence, it is not compared against QCN-Switch proposal evaluated next. In fact, QCN-Switch is initially designed thinking in overcoming the limitations of MAR-bP.

4.7.4 QCN-Switch performance results

This section presents the performance results of the proposed *QCN-Switch: adaptive routing based on ECN messages*, introduced in Section 4.6. First, steady-state traffic is considered in Section 4.7.4.1, measuring throughput, latency and fairness. This first evaluation shows that QCN-Switch with *feedback comparison* probability management variant doing the sampling at the *output queues* exhibits the best results; thus, later sections focus only on that configuration. Next, Section 4.7.4.2 evaluates the response time to traffic changes from *uniform* to *adversarial* traffic patterns, and vice versa. A sensitivity analysis is presented, in Section 4.7.4.3, for the most important parameters of the implementation.

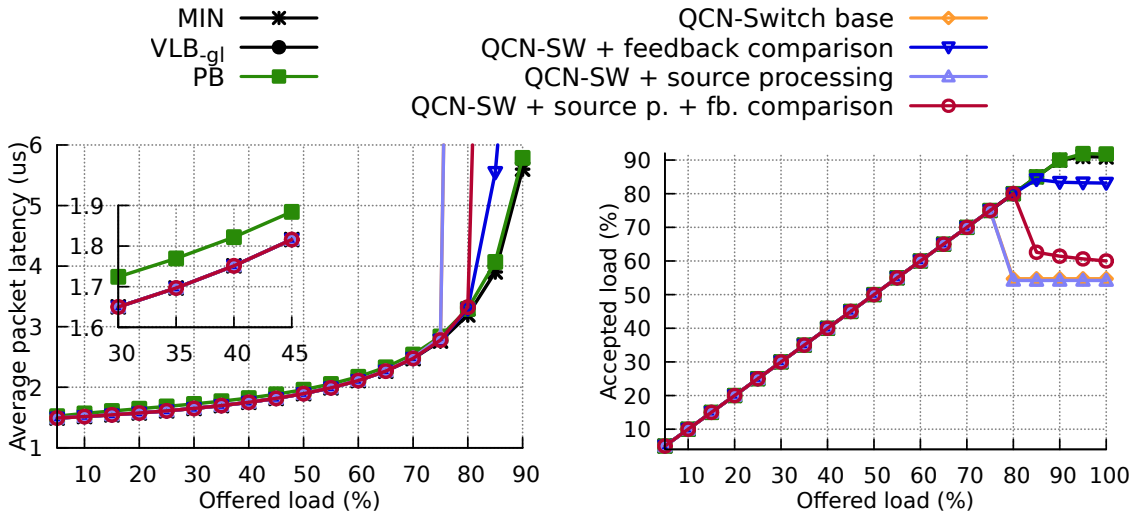
4.7.4.1 Performance under steady loads

This section presents average packet latency and throughput results under *random uniform* and *adversarial shift* with offset $i = 1$ traffic patterns. MIN and VLB_{lgl} routing algorithms are used as the reference routings for UN and ADV+1 traffic patterns respectively. QCN sampling at input and output queues are evaluated separately, considering the proposed variants and mechanisms designed for QCN-Switch.

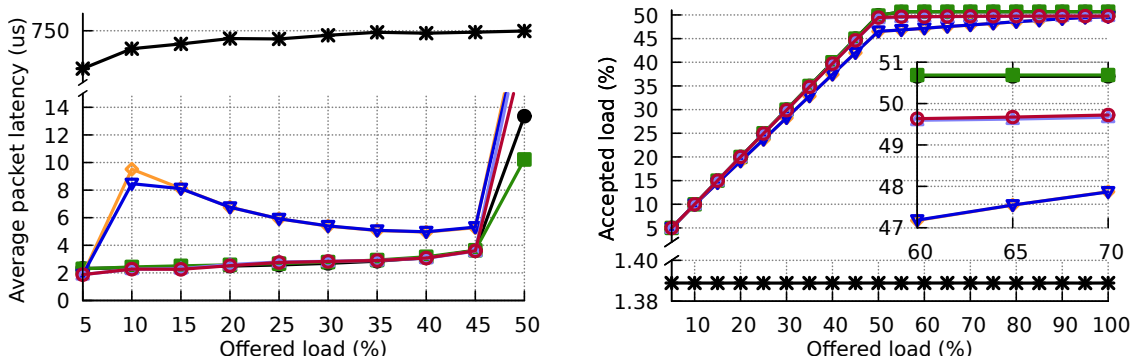
Note that throughput under ADV+1 traffic is limited to $\frac{1}{a \cdot p} = \frac{1}{2h^2} = \frac{1}{72} \simeq 0.01389$ phits/node/cycle when using minimal routing, because only 1 of the 72 global network links per group is used, and to 0.5 phits/node/cycle when using non-minimal routing due to the randomization of traffic which increases the load in the network. This performance limitation under minimal routing is an effect of the topology, not of congestion.

4.7.4.1.1 QCN-Switch sampling at input buffers. Figure 4-15 presents the results when QCN-Switch implements *occupancy sampling* at the *input buffers* under random uniform and adversarial shift traffic patterns. *QCN-Switch base* is the base proposal using the AIMD probability management variant, *QCN-SW + feedback comparison* and *QCN-SW + source processing* implement, respectively, the base proposal of QCN-Switch sampling at input buffers plus the *feedback comparison* probability management variant and the *source processing* mechanism, and *QCN-SW + source p. + fb. comparison* implements QCN-Switch proposal sampling at input buffers plus the *feedback comparison* probability management variant and the *source processing* mechanism.

Under uniform traffic in Figure 4-15(a), all the QCN-based proposals obtain optimal latency, similar to the reference MIN. PB sends part of the traffic non-minimally, which slightly increases its latency, as observed in the inset plot. At high loads, both QCN-Switch base and QCN-SW + source processing decrease their *minimal port probability* values after receiving CNMs, diverting more traffic towards non-minimal paths. Such strategy does not reduce overall congestion; on the contrary, it causes throughput at saturation plunge. However, QCN-SW + feedback comparison corrects this problem and presents good throughput



(a) Average packet latency and throughput under uniform traffic.



(b) Average packet latency and throughput under adversarial traffic.

Figure 4-15: Average packet latency and throughput of QCN-Switch base, feedback comparison probability management variant and source processing mechanism with input-port sampling under (a) UN and (b) ADV+1 traffic patterns.

at saturation; while there is a slight drop, it is relatively small and obtained throughput is competitive with the adaptive reference PB. On the other hand, when feedback comparison is combined with source processing, the throughput loss at saturation reappears.

In the context of ADV+1 in Figure 4-15(b), QCN-Switch base and QCN-SW + feedback comparison have a large latency at low load because most switches are not aware of the congestion caused by the adversarial traffic; indeed, all hosts connected to the respective switch R_{OUT} of each group try to send all traffic following a minimal path. QCN-SW + source processing, alone or combined with feedback comparison eliminates this rise on packet latency, since it allows all switches in a group to detect congestion and to deal with it by using the non-minimal paths. This effect is also visible in Figure 4-16 focusing on throughput fairness. Unfortunately, the latency of QCN-SW + feedback comparison, which was the only competitive one in terms of throughput under even loads as it can be seen in Figure 4-15(a), is worse than QCN-Switch. The comparison with the average feedback value Fb_{avg} makes minimal routing more frequent, which increases average latency under ADV+1.

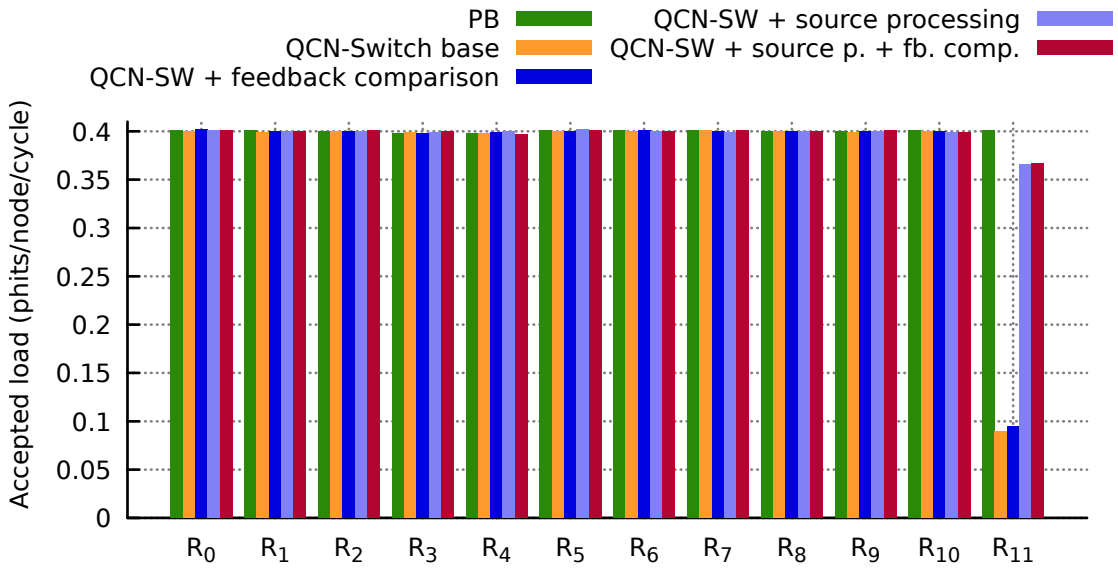


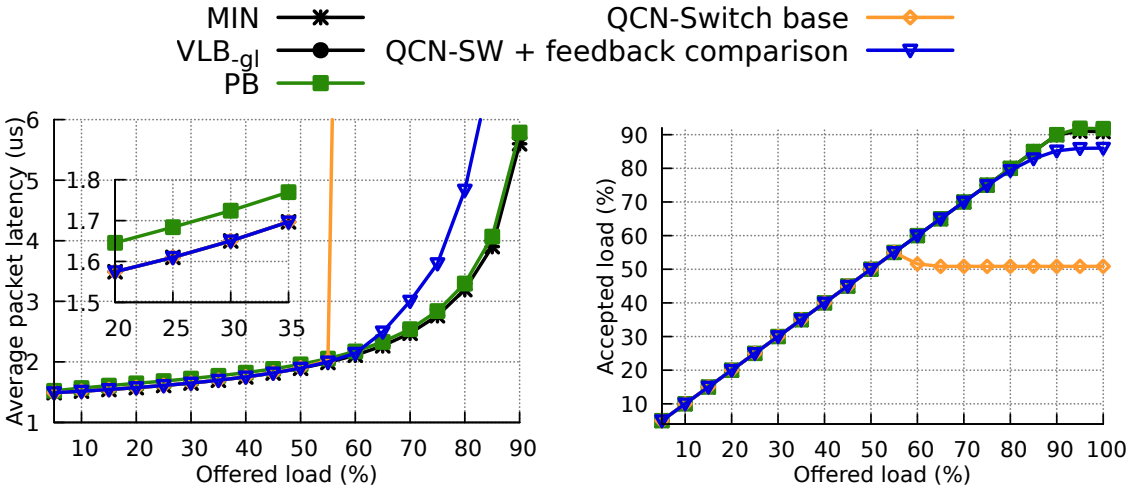
Figure 4-16: Throughput injected in each switch of G_0 for the QCN-Switch base, feedback comparison probability management variant and source processing mechanism with input-port sampling under ADV+1 traffic pattern with a load of 0.4 phits/node/cycle.

Throughput results under ADV+1 are consistent with the latency values analyzed above. QCN-SW + source processing, alone or with feedback comparison obtains the maximum saturation throughput, and despite suffering some congestion at high loads, its throughput is competitive with the oblivious Valiant. Note that both QCN-Switch base and QCN-SW + feedback comparison have a minor loss of throughput, around 2%, which occurs below the saturation point. This is observed easily in the slope of their throughput curves before saturation, lower than the expected 45° and this is indicative of throughput unfairness issues. Figure 4-16 explores this problem in more detail, comparing the throughput obtained by each switch of Group 0 at load 0.4 phits/node/cycle. Switch R_{11} corresponds to the router R_{OUT} in Figure 3-5. Using QCN-Switch base, R_{11} injects significantly less traffic than other switches of the group. As discussed in Section 4.6, R_{OUT} does not receive a significant number of CNMs from the global link, because all traffic received in the destination group is quickly dispatched to its target switch, so the queues do not get full; recall that during the current evaluation subsection the detection point is implemented at the input buffers. As expected, this unfairness is resolved when source processing is used, as R_{11} sends to itself the same notification that would be sent to other switch when congestion is detected, which means R_{11} also changes its *minimal port probability* for congested ports.

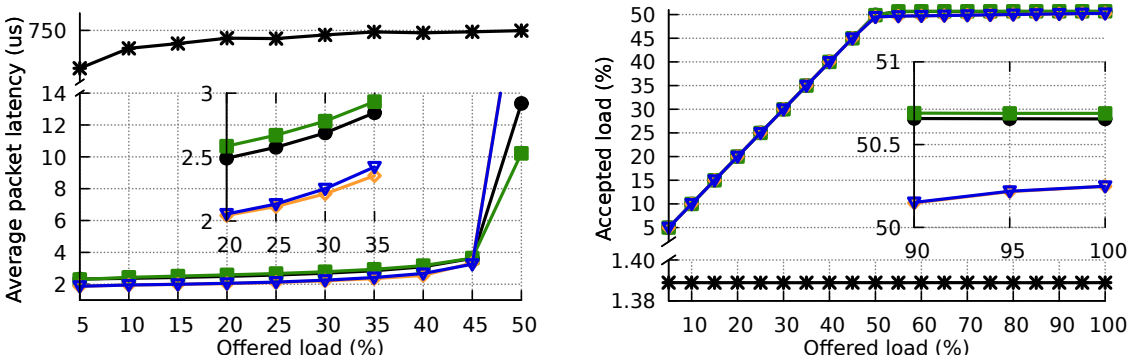
4.7.4.1.2 QCN-Switch sampling at output buffers. Figure 4-17 shows average packet latency and throughput under uniform and adversarial shift traffic patterns[13] using *occupancy sampling* at the *output queues* with the AIMD probability management variant (*QCN-Switch*

[13] Alternative traffic patterns, which are not adversarial but impose significant network congestion, were evaluated in [25] and are omitted here by concision and to keep the discourse in line with the whole dissertation.

base), versus the reference routing algorithms. Note the *source-processing* mechanism is not necessary because the base case is already fair when using output-port sampling. *QCN-SW + feedback comparison* represents QCN-Switch using occupancy sampling at the output buffers combined with the *feedback comparison* probability management variant.



(a) Average packet latency and throughput under uniform traffic.



(b) Average packet latency and throughput under adversarial traffic.

Figure 4-17: Average packet latency and throughput of the QCN-Switch base and feedback comparison probability management variant with output-buffer sampling under (a) UN and (b) ADV+1 traffic patterns.

Under uniform traffic, QCN-Switch base suffers a lost of peak throughput which is due to the "positive control loop" described in Section 4.6.3, which starts when congestion is detected and results on most traffic using non-minimally paths. However, QCN-SW + feedback comparison corrects this problem and presents good throughput at saturation, which is competitive with the adaptive reference PB. At medium loads, PB sends part of the traffic non-minimally, which increases latency as shown in the inset plot; by contrast, the two QCN-Switch proposals obtain optimal latency, similar to the reference MIN.

In the context of adversarial traffic, the base proposal QCN-Switch base reduces network latency over PB reference because it eliminates the local hop on the intermediate group. Latency results are also better than when using input-port sampling in Figure 4-15(b), because

congestion detection is more fair in this case. Throughput results under adversarial traffic are consistent with the latency values analyzed above, resulting in almost ideal throughput just below the reference mechanisms.

Figure 4-18 presents throughput fairness results with sampling at output queues. Output-port sampling allows to generate CNMs that are intercepted by any switch in the group, including the same router that generates the message. For this reason, the base proposal is fair and there is no need to implement *source processing* mechanism. Adding *feedback comparison* probability management variant to the base case does not impact on its fairness.

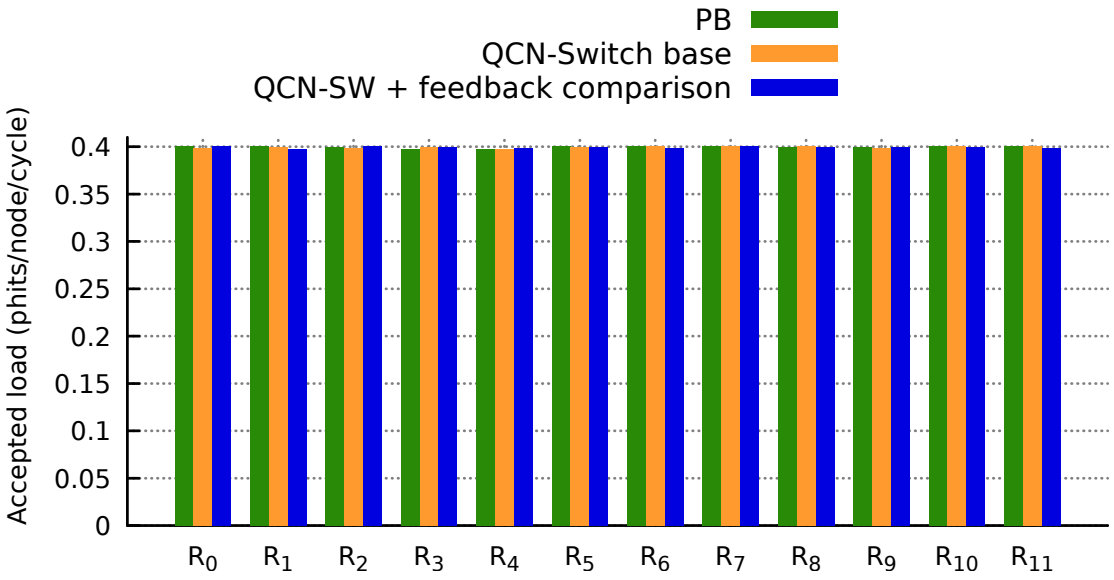


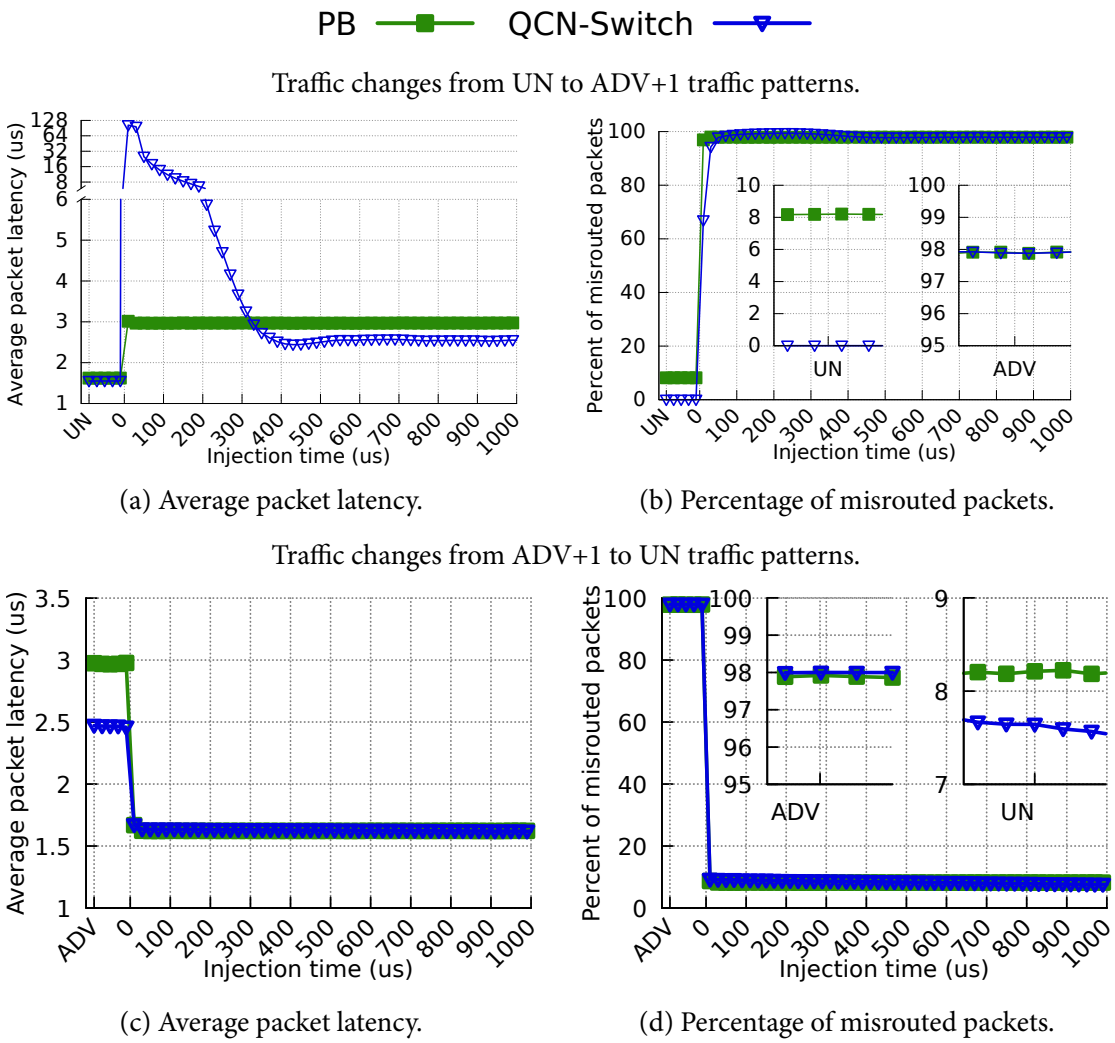
Figure 4-18: Throughput injected in each switch of G_0 for the QCN-Switch base and feedback comparison probability management variant with output-buffer sampling under ADV+1 traffic pattern with a load of 0.4 phits/node/cycle.

Although the results exposed by Neeser *et al.* [142] suggest input-port sampling to perform adaptive routing based on QCN, the evaluation results exposed in this section show that QCN-Switch achieves better performance and more consistent results sampling at the output buffers than at input queues. Moreover, *feedback comparison* improves throughput. For this reason, the rest of the evaluation is focused only on this configuration. In upcoming sections all references to the proposed routing algorithm are denoted simply as *QCN-Switch* to refer to QCN-Switch using *occupancy sampling* at the *output buffers* combined with the *feedback comparison* probability management variant.

4.7.4.2 Performance under transient loads

This section evaluates the response time to traffic changes by modeling a network load of 0.4 phits/node/cycle that changes from *uniform* to *adversarial* patterns and vice versa. The network is warmed-up for 2.4 ms with the first traffic pattern. At this point, denoted as $t = 0$, the network is stable and then it is changed to the second traffic pattern and run this

traffic pattern for 100 μs. During this period, the evolution of the latency and misrouted packets are registered, showing how the network transitions from one pattern to the other. Figure 4-19 presents the average packet latency and the amount of misrouted packets for both transient loads, UN to ADV+1 and vice versa.



(a) Average packet latency.

(b) Percentage of misrouted packets.

(c) Average packet latency.

(d) Percentage of misrouted packets.

Figure 4-19: Average packet latency and percentage of misrouted packets of PB and the QCN-Switch with output-buffer sampling and feedback-comparison probability management variant changing from UN to ADV+1 traffic patterns at injection time *t=0* and vice versa.

The latency of QCN-Switch proposal under ADV+1 is lower than PB; this result is consistent with Figure 4-17. The response time from UN to ADV+1 is almost 0.4 ms, while from ADV+1 to UN it is much quicker (less than 0.005 ms). This change is faster because UN uses all ports evenly and there is only one rule on each switch, i.e., the one associated with the global link, that needs updating to use minimal paths. In other words, sending a small amount of traffic non-minimally under UN has minimal impact on performance. On the other hand, when the traffic changes from UN to ADV+1, all the traffic in a switch is funneled towards the same global link until those rules are changed to use non-minimal paths, creating a temporary hot-spot that requires some time to be absorbed by the network.

Comparing the percentage of misrouted packets between Figures 4-19(b) and 4-19(d) under the uniform traffic phase, it can be seen that they are not the same; in the first plot the switch has not detected any congestion during the UN traffic phase, so there is 0% misrouting; on the second plot the percentage of traffic sent minimally is still trending down after the previous ADV+1 traffic phase, and although it initially drops very quickly from 100% to 8%, it progresses very slowly towards 0 afterwards.

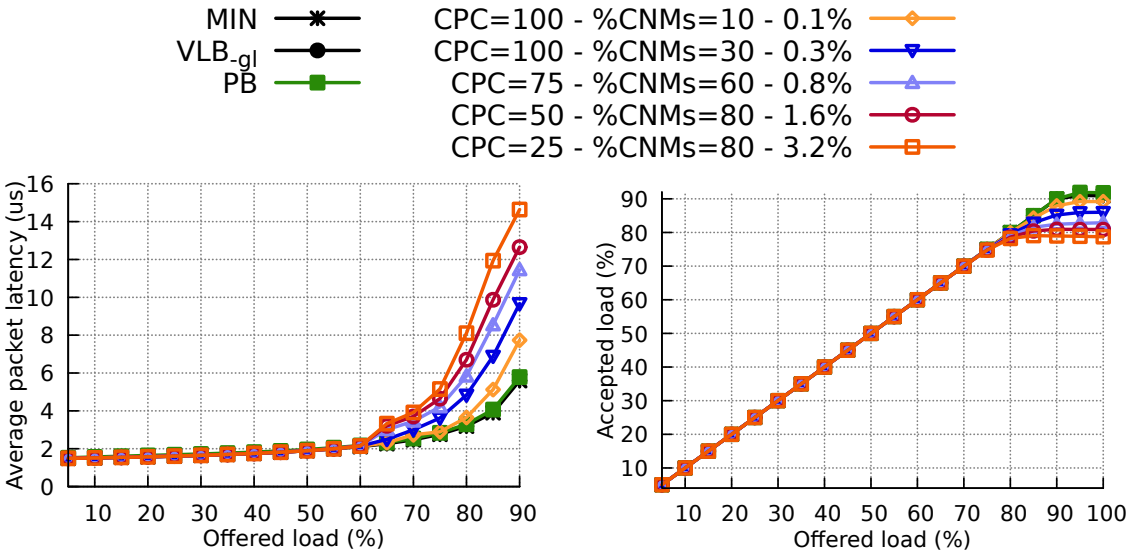
4.7.4.3 Sensitivity analysis

This section studies the impact of the different parameters used in the routing algorithm, both from the QCN notification part and the proposed implementation in the receiving switches. Both steady-state and transient traffics have been used, changing one parameter at a time from the configuration employed in previous evaluations. Hereon, the QCN-Switch evaluated employs the *QCN-Switch + feedback comparison* model using occupancy sampling in the output buffers, which represents the best configuration as shown in previous sections. Finally, the selected parameters set is evaluated for different network sizes.

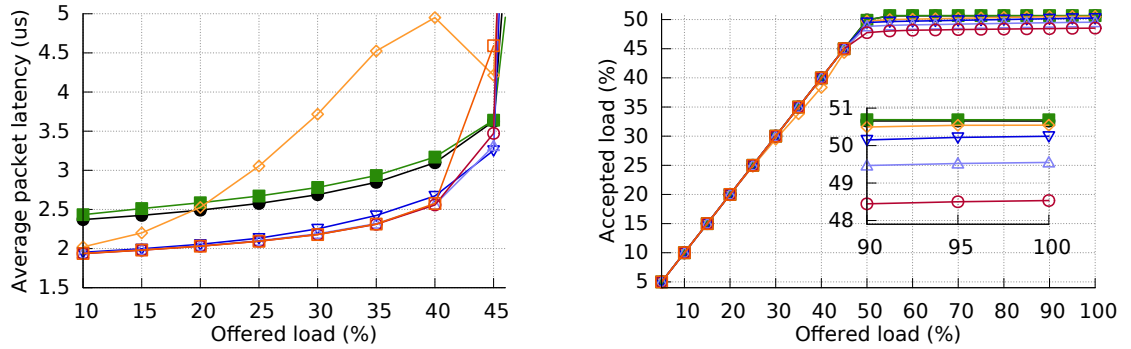
4.7.4.3.1 Number of notifications: *CPC* and *%CNMs*. The *CPC* parameter represents the frequency of buffer sampling in QCN CPs. This section considers values for this parameter to sample from every 100 packets, like QCN standard value, up to sample every 25 packets. The parameter PC_l, which defines the amount of packets sent following a given routing table entry before an increasing of its output *minimal port probability*, is set accordingly.

Similarly, %*CNMs* is a QCN parameter that indicates how many CNMs are generated from buffer sampling operations with a negative result which implies congestion. By default, %*CNMs* is 10% in QCN, this is, only one message is sent every ten sampling operations which report a negative result. However, evaluations in previous sections employ %*CNMs* = 30%, and in this section considers values ranging from 10% to 80%. Combining *CPC* and %*CNMs* parameters, this section evaluates a set of configurations that generates from 1 to 32 CNMs per 1,000 packets forwarded, denoted as 0.1% to 3.2% in the legend.

Figure 4-20 presents average latency and throughput under UN and ADV+1 traffics. Figure 4-20(a) shows a trade-off between the amount of messages generated and the performance at UN loads. The most aggressive configuration (3.2%), and similar ones, suffer at high loads because the overhead of the CNMs causes some non-minimal forwarding which increases latency. In general, increasing the amount of CNMs has a slight impact on saturation throughput under both traffic patterns. Besides, it increases latency at high loads under UN traffic. The default configuration for QCN (0.1%) is too low for the proposed routing, as it produces high latency at medium loads under ADV+1 traffic as seen in Figure 4-20(b). Overall, the selected configuration (0.3%) produces competitive performance under UN and ADV+1 traffics, both in terms of low latency at medium loads and peak throughput.



(a) Average packet latency and throughput under UN traffic pattern.

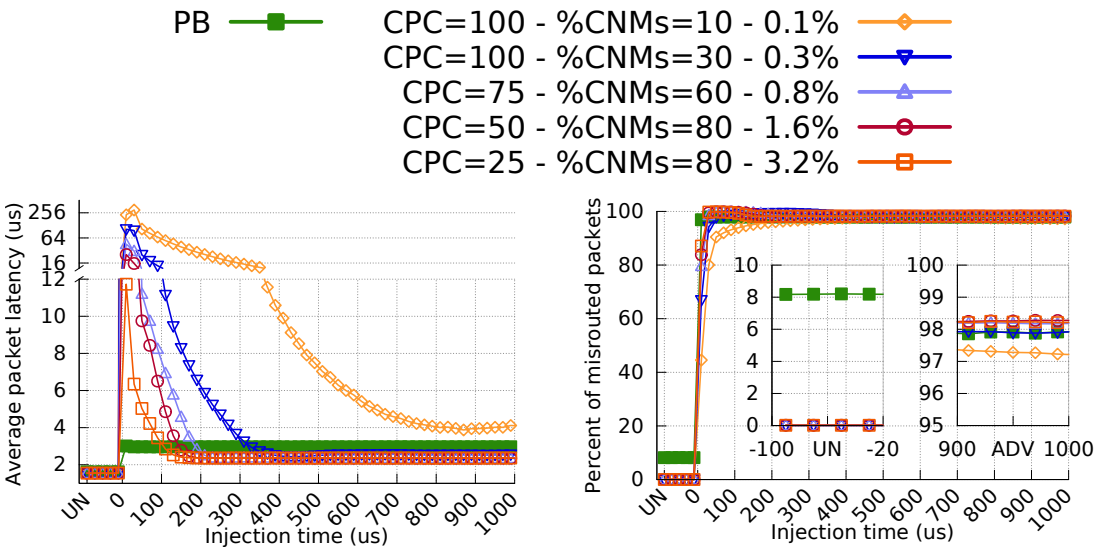


(b) Average packet latency and throughput under ADV+1 traffic pattern.

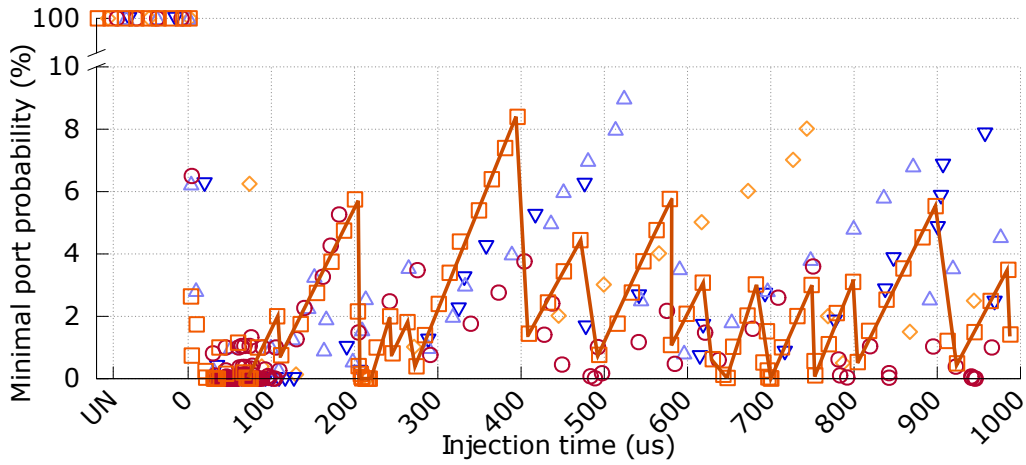
Figure 4-20: Average packet latency and throughput of the QCN-Switch with output-buffer sampling and feedback comparison, for different values of sampling interval and percentage of CNMs sent under (a) UN and (b) ADV+1 traffic patterns.

Figure 4-21(a) shows the evolution of both average packet latency and percentage of misrouted packets when the traffic pattern changes from uniform to adversarial shift. As expected, increasing the frequency of congestion notification messages has the greatest impact on the response time to traffic changes: the switches are aware of the changes on the network load sooner as more CNMs are received. Response time is considered as the time spent from the traffic change, at time $t = 0$, to the moment in which average latency approximately converges to the new value. Using the default parameters from QCN presents a large response time, close to 1 ms. Using the base configuration presented in Table 4-3, a response time of 0.4 *ms* is obtained, and increasing the amount of messages continues decreasing the response time up to near 0.1 *ms*. The CNM frequency has no impact on the percentage of misrouted packets, except a minor effect at the start of the transition phase.

Finally, Figure 4-21(b) shows the evolution of the *minimal port probability* of the port connected to R_{OUT} of a switch in the middle of group zero, during a traffic pattern change



(a) Average packet latency and percentage of misrouted packets.



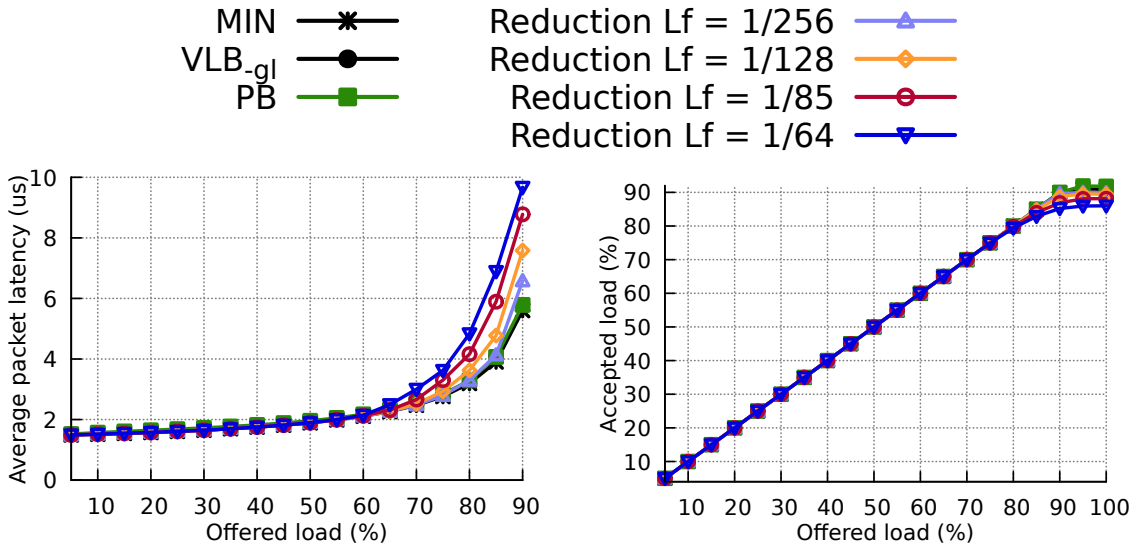
(b) Evolution of the percentage of minimal routing in an intermediate router in G_0, measured in the congested port that connects minimally to the destination in ADV+1. Each marker represents one change in the minimal port probability.

Figure 4-21: Transient response when traffic changes from UN to ADV+1, for different values of sampling interval and percentage of CNMs.

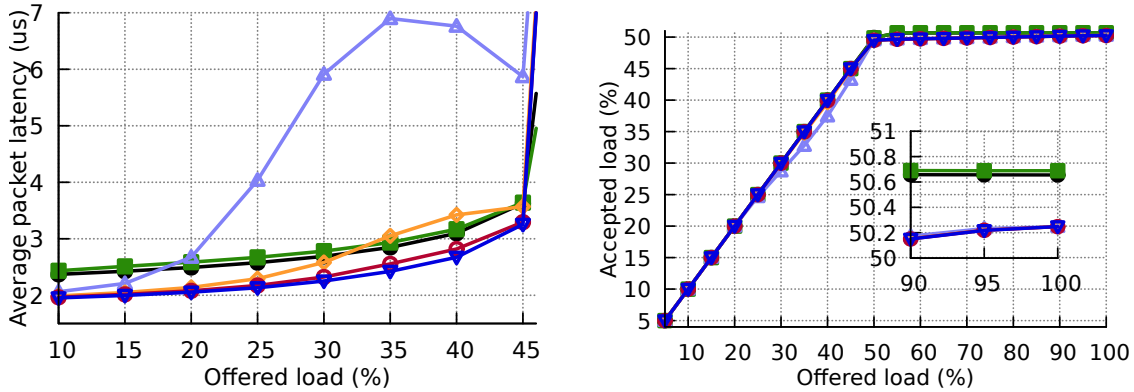
from UN to ADV+1. For each configuration analyzed, which are codenamed in the legend, one marker is presented in each instant in which the port probability value is updated. All configurations reduce the minimal port probability rather quickly after the traffic change, and it remains relatively low, under 10%. However, there is a significant variability in the probability values. The evolution of the minimal port probability follows a sawtooth pattern caused by the AIMD policy used. The increase stairs are steeper in the configurations with more notifications (such as the square orange marker - 3.2%, whose evolution is highlighted) because a more aggressive *CPC* value also implies a shorter interval PC_I between probability increases. However, the results in Figure 4-21(b) do not present any clear conclusive difference, as it is observed later in the similar plot in Figure 4-23(a) for the parameter L_f.

In conclusion, the selection of the parameters *CPC* and *%CNMs* presents a trade-off between latency under high uniform traffic loads and reaction time needed to adapt to traffic changes. The default values used in Table 4-3 present a competitive latency and a sufficiently quick response time, according to the analysis of HPC applications requirements in Section 4.2.

4.7.4.3.2 Reduction limiting factor L_f. It determines how strongly the switch reacts when receiving CNM by decreasing the probability of sending traffic minimally. Figure 4-22 presents steady-state results under uniform and adversarial shift traffic patterns for values of $L_f = \{\frac{1}{256}, \frac{1}{128}, \frac{1}{85}, \frac{1}{64}\}$, which implies approximately a maximum probability reduction of 0.25, 0.5, 0.75 and 1.



(a) Average packet latency and throughput under uniform traffic.

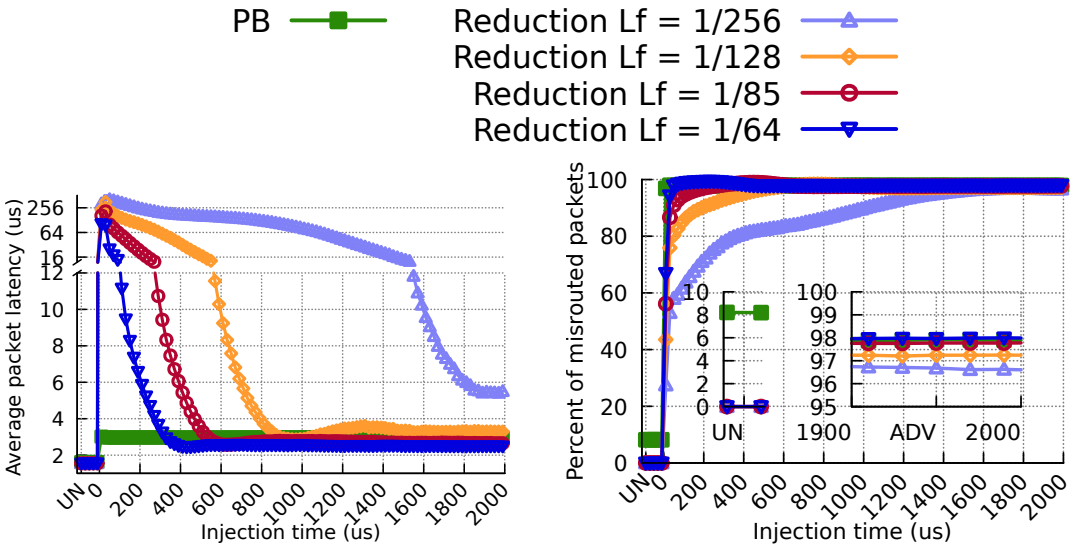


(b) Average packet latency and throughput under adversarial traffic.

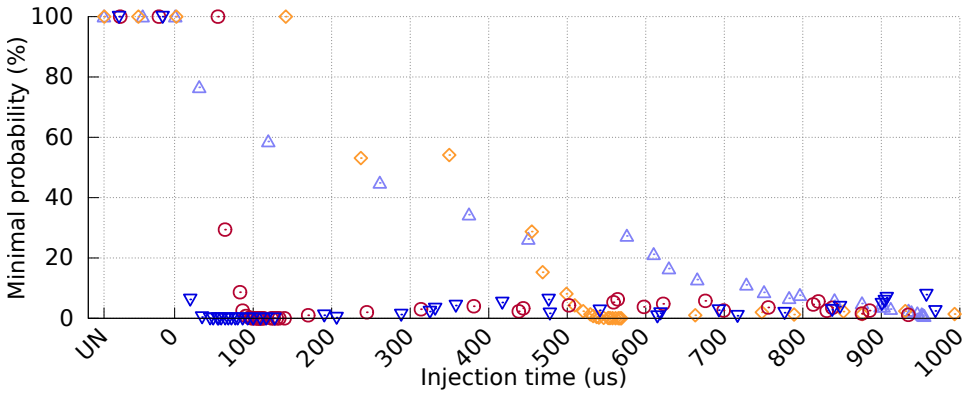
Figure 4-22: Average packet latency and throughput for different limiting factors L_f under UN and ADV+1 traffic patterns.

The default value in Table 4-3, $L_f = \frac{1}{64}$, is the most aggressive one and it produces the lowest latency under ADV+1 traffic. However, other values present better latency under uniform traffic at high loads. The impact on throughput is very small in both cases.

Figure 4-23 presents results under transient traffic pattern. Response time is affected considerably by the value of L_f. The base value $L_f = \frac{1}{64}$ corresponds to a maximum reduction factor of $R = 1 - \frac{63}{64}$, this is, it can reduce the port percentage maximally after receiving a single CNM. Larger values of L_f present a slower convergence time, which is particularly obvious in the average latency in Figure 4-23(a) and port probability in Figure 4-23(b). In particular, $L_f = \frac{1}{128}$, which corresponds to a maximum reduction value of $R \approx 50\%$, is often employed in control mechanisms based on an AIMD policy, such as TCP or QCN. It can be observed that using such value causes significantly slower convergence: it requires about six probability updates and more than twice the time as the base configuration to converge.



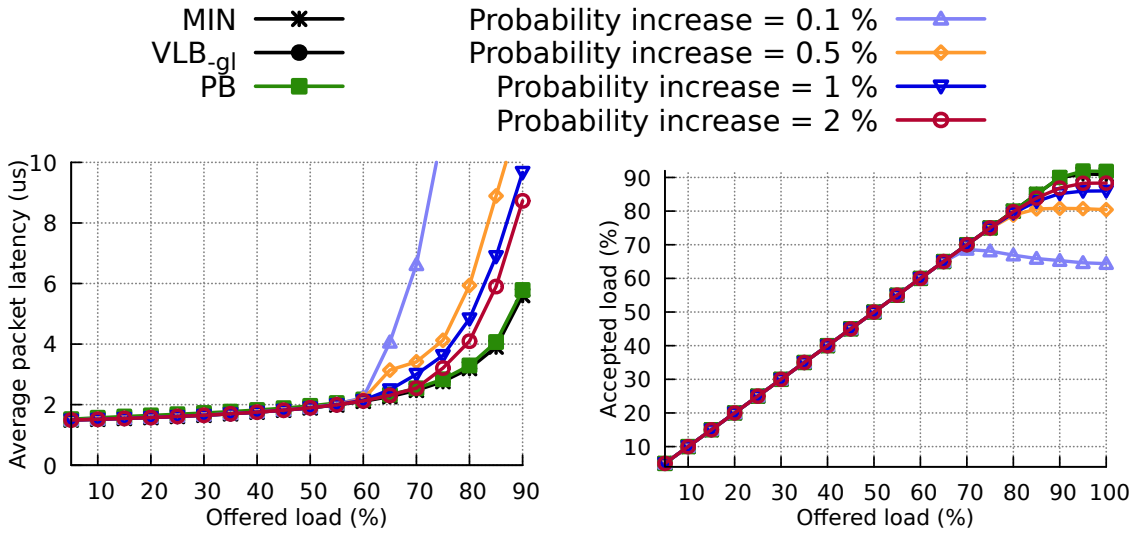
(a) Average packet latency and percentage of misrouted packets.



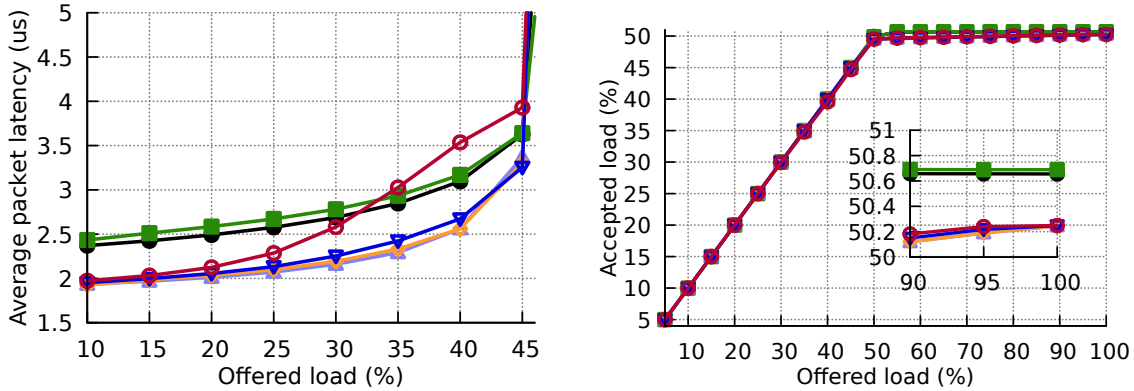
(b) Evolution of the percentage of minimal routing in an intermediate router in group 0, measured in the congested port that connects minimally to the destination in ADV+1. Each marker represents one change in the probability.

Figure 4-23: Transient response when traffic changes from UN to ADV+1 traffic pattern, for different values of the reduction limiting factor L_f.

4.7.4.3.3 Probability Increase *PI*. It determines the increase of the *minimal port probability*, when no congestion notifications are received during an interval of PC_I = 100 frames. Figure 4-24 presents results for steady-state traffic for different values of *PI*. With low values for *PI* (0.1% or 0.5%), throughput and latency results under uniform traffic at high loads, presented in Figure 4-24(a), are poor because the system is slow to reduce non-minimal routing after spurious CNMs are received. By contrast, high values of *PI* increase latency under adversarial shift traffic in Figure 4-24(b), because a brief absence of notifications causes the system to revert to minimal routing too soon.



(a) Average packet latency and throughput under UN traffic pattern.

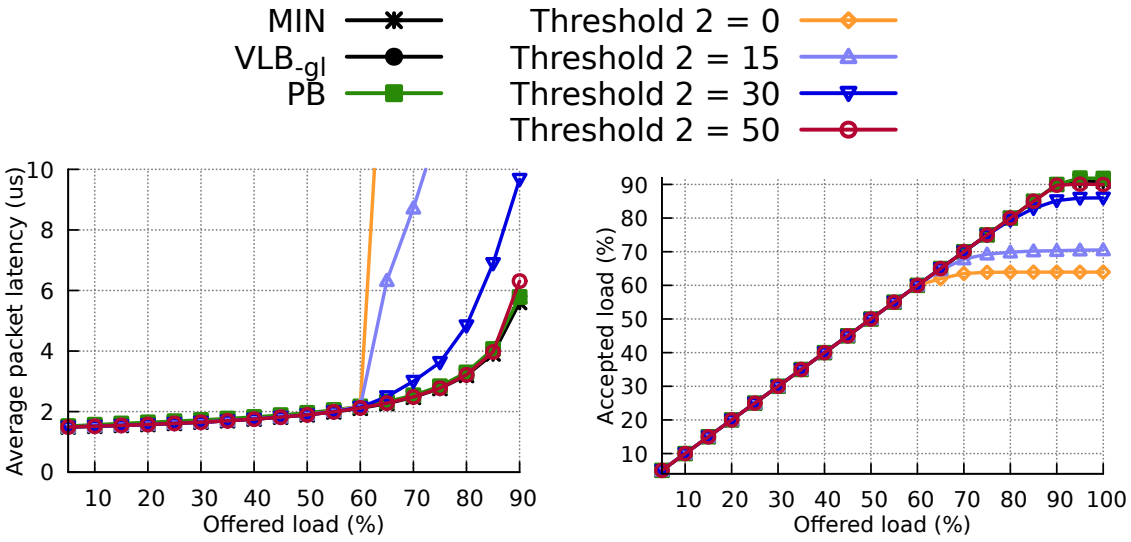


(b) Average packet latency and throughput under ADV+1 traffic pattern.

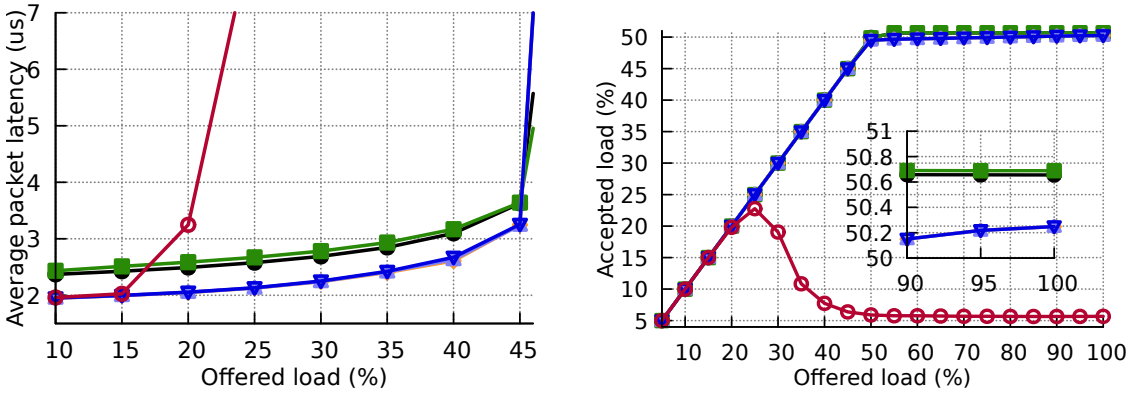
Figure 4-24: Average packet latency and throughput for different values of probability increase *PI* under UN and ADV+1 traffic patterns.

Obviously, this parameter also determines the convergence time when transitioning from ADV+1 to UN traffic. However, as discussed in Section 4.7.1.2, this transition time is not as critical as the change from UN to ADV+1, and Figures 4-19(c) and 4-19(d) already presented an acceptable transient result. Considering this reason, transient results are omitted for brevity, but it is clear that a higher *PI* would converge to 0% misrouted packets faster.

4.7.4.3.4 Feedback comparison thresholds Th_1 and Th_2. Section 4.6.3 introduces the use of two thresholds to avoid excessive variations in the *minimal port probability* of the ports. Threshold Th_1 controls the range of feedback values with which the probability is increased when no CNM is received; preliminary evaluations show that it has a negligible impact on performance. For this reason, this section presents an evaluation of the impact of threshold Th_2, which restricts the reduction of the minimal probabilities of each port. This threshold represents the difference required between the feedback value received in a CNM, which tops at 63, and the average Fb_{avg}, in order to reduce the minimal port probability. Figure 4-25 presents an evaluation of the impact of this threshold on steady load, for different values of Th_2 from 0 to 50.



(a) Average packet latency and throughput under UN traffic pattern.



(b) Average packet latency and throughput under ADV+1 traffic pattern.

Figure 4-25: Average packet latency and throughput for different feedback-comparison threshold Th_2 under UN and ADV+1 traffic patterns.

Figure 4-25(a) presents average packet latency and throughput under uniform traffic. A small to medium value Th_2 allows the switch to ignore transient congestion fluctuations. A large value means that only CNMs that report an abrupt change will cause a reduction of

the minimal port probability, which is good for UN traffic. In fact, not using a threshold (i.e., $Th_2 = 0$) produces poor latency and throughput. However, a very high value is bad under adversarial shift traffic presented in Figure 4-25(b) because probability is not properly reduced and throughput goes down. Thus, using the intermediate value of $Th_2 = 30$ is the best option for a network that may support different traffic loads.

4.7.4.3.5 Network size. Previous sections have evaluated the parameter selection using a network size of 5,256 terminals, which corresponds to a Dragonfly network using $h = 6$ global links per switches and 23 ports used per router. Figure 4-26 shows the results for different network sizes, ranging from 1,056 ($h = 4$, 15 ports used per switch) to 16,512 compute hosts ($h = 8$, 31 ports used per switch) under random uniform and adversarial shift traffic patterns. Evaluations show maximum throughput in saturation, and latency at intermediate loads. Except for the aforementioned network size, all the evaluations use the same set of parameters presented in Table 4-3; *QCN-Switch* parameters are not modified for different network sizes. MIN and VLB$_{-gl}$ oblivious routings are also included as a reference.

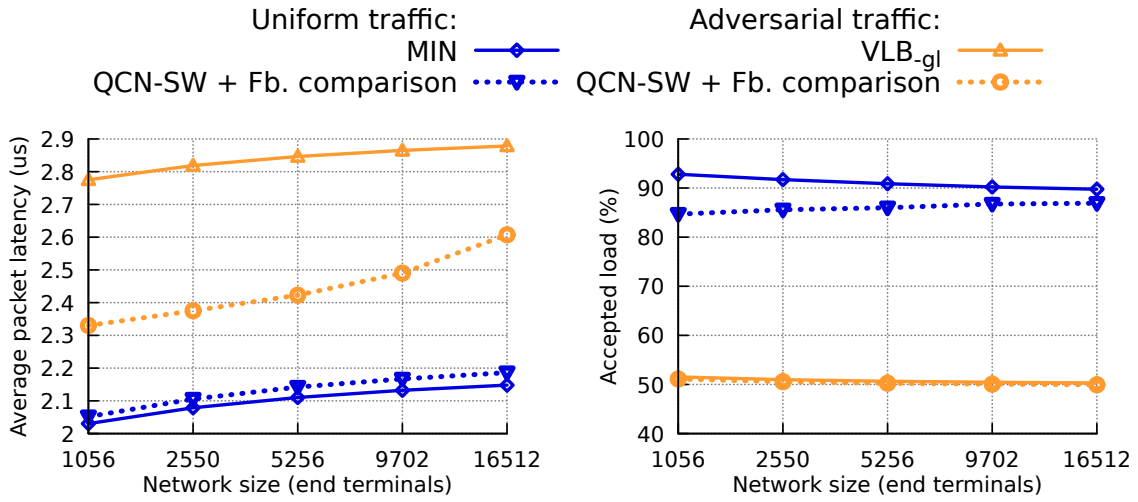


Figure 4-26: Average packet latency and throughput for different network sizes under UN (blue) and ADV+1 (yellow) traffic patterns. Average packet latency is obtained under 65% (UN) or 35% (ADV+1) load and throughput is obtained under 100% of load. Results for oblivious routing (MIN and VLB) are also presented as a reference.

Saturation throughput remains fairly stable for different network sizes. Under ADV+1 traffic *QCN-Switch* saturation throughput is in all cases virtually identical to Valiant. Under UN traffic, it is close to the reference MIN, and it is better for larger network sizes. Average packet latency slightly grows with the network size in both oblivious references. A similar effect occurs for *QCN-Switch*, but latency increases slightly more under ADV+1 traffic.

These results suggest that the selected parameters are valid for a broad range of network sizes, but they accept a fine-tuning for far different configurations.

4.8 Conclusions

This chapter identifies the requirements of high-performance computing interconnection networks compared to traditional data center ones. Also, it explores proposals designed for scalable DC networks using commodity switches in exascale-level HPC interconnects. The results suggest that most of the proposals do not properly apply to HPC systems, such as network overlay technologies VXLAN or TRILL that require large switch tables relying on flooding; on-the-fly MAC address rewriting mechanisms which interfere with layer-2 service announcement; or fine-grained per-flow load balancing mechanisms.

By contrast, the proposals of this chapter rely on a hierarchical routing based on location-dependent MAC addresses, TCAM rules compaction, a dynamic mechanism to assign location-dependent MAC addresses to compute hosts and conditional flow rules to implement the proposed source-adaptive routing algorithms in the network devices. The implementation is realistic and requires minimal changes in OpenFlow switches. This set of mechanisms permit the implementation of low-power networks based on Dragonflies and Flattened Butterflies topologies using commodity Ethernet switches. Accomplished evaluations show that performance is optimal under random uniform traffic pattern and remains competitive against ad-hoc HPC routing algorithms under adversarial shift traffic pattern, while providing energy savings up to 54%.

Two routing algorithms have been proposed based on *conditional flow rules*, one based on *pause* and another based on *explicit congestion notification* messages. The former one is initially proposed to overcome the lack of credits on Ethernet networks, and is a naïve implementation. This simple implementation experiences performance limitations, such as high latency, and throughput drops which have been overcome with the latter proposal based on ECN messages.

The proposed *QCN-Switch* uses statistical routing driven by the interception of explicit congestion notification messages generated using commodity QCN. Preliminary *QCN-Switch* design exploration shows that it is better to rely on output-port sampling than on input-port sampling, because it not only provides better performance but it is critical to provide fairness under adversarial traffic. *Feedback comparison* probability management variant is key to eliminate costly misrouting that occurs at saturated benign loads, while still using non-minimal paths to reduce any other congestion caused by uneven loads.

The steady-state performance of the final *QCN-Switch* design is comparable to state-of-the-art sophisticated high-performance alternatives such as Piggyback. Considering transient changes of traffic, while the resulting design does not adapt routing as quickly as other routing algorithms that rely on credit counters, it responds in a sub-millisecond time frame, which is typically enough for most applications. A sensitivity analysis presents the design trade-offs, particularly improving performance for uniform or adversarial traffic patterns or trading off faster response time for lower saturation throughput.

In conclusion, the trade-offs of relying exclusively on ECN notifications for non-minimal adaptive routing has been explored, and a feasible routing and a switch design for HPC low-diameter networks based on commodity Ethernet technology have been also provided.

Non-minimal Adaptive Routing With Latency Improvements

5

Low-diameter high-radix interconnection networks require non-minimal adaptive routing algorithms to avoid network congestion. These routings dynamically select between forwarding traffic following a minimal or a non-minimal path. To accomplish that, they need to define two key aspects: 1) how to select between both types of paths and 2) which non-minimal path to employ. The first typically relies on the *Universal Globally Adaptive Load-balancing* routing algorithm (UGAL, [172]) but it can also be based on the *QCN-Switch* architecture presented in the previous chapter. The second typically relies on *Valiant Load-Balancing* routing algorithm (VLB, [187]), which diverts traffic to a random intermediate router before sending it to its final destination.

This chapter is motivated, in Section 5.1, by an analysis of the original VLB that suggests two potential improvements regarding the selection of the intermediate node R_{ROOT}. First, when traffic is local[1] the randomization introduced by VLB results in unnecessarily long paths. Instead, a *restricted* version of VLB, introduced in Section 5.2, randomizes traffic within a *local partition*, avoiding congestion and generating shorter paths. Second, in certain cases the path to the selected R_{ROOT} may be blocked. To overcome this situation, a version of VLB with *recomputation*, introduced in Section 5.3, selects a new R_{ROOT} on each *routing computation* stage as long as the associated path to reach the currently selected one remains stalled. The main proposal of this chapter, introduced in Section 5.4, is *ACOR: Adaptive Congestion-Oblivious Routing*. It leverages the *restricted* and *recomputation* techniques to reduce path length for local and global traffic, and to extend it when the network conditions are adverse. Same as VLB, ACOR does not send traffic minimally. However, it can be combined with non-minimal adaptive routing mechanisms such as Piggyback. In this case, Piggyback selects between minimal and non-minimal paths and ACOR determines the non-minimal path.

Evaluation results, in Section 5.5, show that *restricted* improvement of VLB is highly effective in cases of local traffic and VLB with *recomputation* increases injection, further reducing average latency and increasing throughput. ACOR avoids any pathological throughput limitation and reduces base latency in all cases, up to 28% standalone and up to 25.5% when combined with Piggyback, while requiring a simple implementation. This combination achieves rivaling throughput and significantly lower latency compared to base Piggyback.

[1] Traffic is local to a particular partition of the network which must be defined for each topology. For example, this partition can be the first dimension in a Flattened Butterfly or the group in a Dragonfly.

Chapter contents

5.1 Motivation

Under adversarial traffic patterns, presented in Section 3.2.2, *Minimal* routing algorithm (MIN), presented in Section 2.3.1, saturates some network links, which become a bottleneck. VLB routing algorithm, presented in Section 2.3.2, avoids such bottlenecks by randomizing traffic. Original VLB routing first sends traffic minimally to a random intermediate router R_{ROOT} in its *phase A* following a *lgl* path. Then, in its phase B, traffic is sent minimally from R_{ROOT} to the actual destination.

While VLB prevents network bottlenecks, it employs paths that may be unnecessarily long and imposes an excessive load on the network. Short non-minimal paths have been employed in previous works, such as the *lg*- phase A path used in [108, 96]. However, under certain adversarial traffic patterns (e.g., ADV+*h* defined in Section 3.2.2.1.2), it does not randomize traffic enough and some local links in the intermediate groups become a bottleneck.

Topology-custom UGAL routing on Dragonfly (T-UGAL, [156]) pre-computes a set of Valiant paths that reduce average distance on Dragonfly networks with more than one link between each pair of groups, usually denoted as *global trunking*. However, it does not react to the current level of congestion, and so using shorter VLB paths only when congestion is not severe; instead, the precomputed set of paths is used for all traffic loads.

The key goal of this chapter is to find a routing algorithm that employs short non-minimal paths and uses them during the load range in which pathological performance limitations are not introduced, and that dynamically increases the number of hops in VLB phase A when shorter paths introduce those performance issues.

5.2 RVLB: Restricted Valiant Load-Balancing

The original definition of Valiant load-balancing, described in Section 2.3.2, randomizes perfectly the paths of packets in a hypercube network, balancing the use of resources regardless of the traffic pattern. However, it does not consider the case of hierarchical networks or topologies consisting of multiple orthogonal dimensions, where the source and destination may be located in the same partition or subset of dimensions. Hence, selecting a random destination among *all* the switches of the network may lead to unnecessary packet diverting and increased path length for certain topologies. For example, this may occur when the network is hierarchical, such as the Dragonfly or multilevel Fat-tree, or consists of multiple orthogonal dimensions, such as the HyperX network, and both the source and destination are confined to the same partition of the hierarchy (e.g., group, *pod*) or the same subset of dimensions. In such cases, selecting a random destination outside the partition implies leaving and returning to the original partition, which increases the path length without contributing to congestion avoidance.

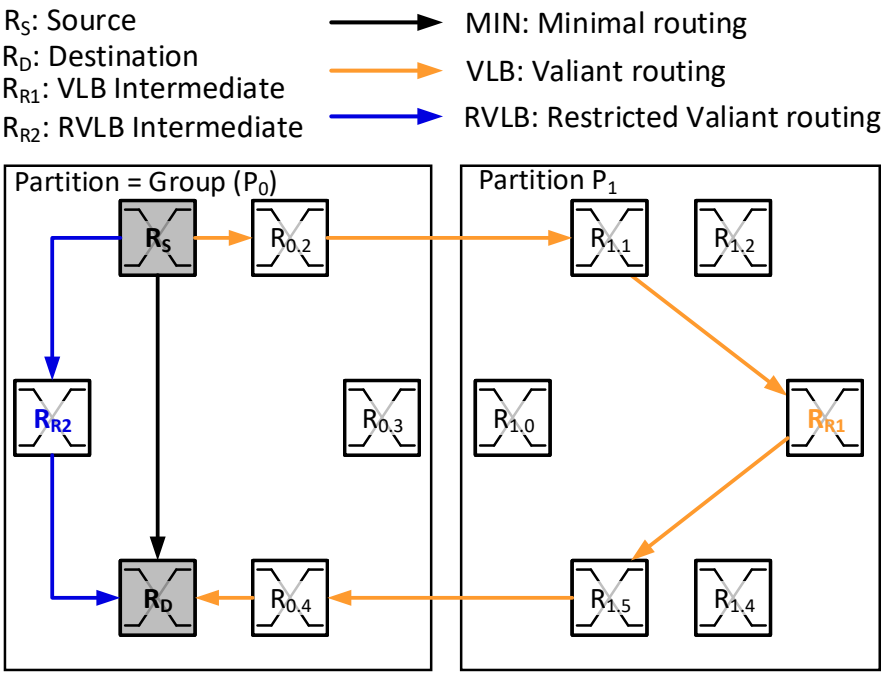
This issue represents a more general case of the *turn-around problem* identified by Yébenes *et al.* [193] in the SlimFly topology, where packets visit the same switch twice in their path, turning around and going back through the same route. In such cases, the subsegment of the path between the two visits to the same switch can be omitted without negatively impacting the benefits of packet randomization. However, in the general case the problem is slightly different, because the paths of VLB phase A, i.e., towards the intermediate switch R_{ROOT}, and phase B, i.e., from $_{ROOT}$ to the actual destination, may not overlap while the complete route is still unnecessarily long. Figure 5-1 depicts this issue in a Dragonfly network with *global trunking*, this is, more than one global link connecting pairs of groups, and in a 4×4 Flattened Butterfly (HyperX). In the Dragonfly using VLB routing algorithm the selected random intermediate switch, R_{R1} in the presented example, is in a remote group, and no switch is traversed twice because the two paths to the intermediate switch and to the destination are disjoint. However, the resultant path is needlessly long; restricting the selection of the intermediate switch to a router local group (R_{R2}), by the proposed *Restricted Valiant Load-Balancing* (RVLB), avoids the congestion issue present in the *minimal* link while providing a shorter path. A similar case occurs in the Flattened Butterfly, where RVLB avoids changing the row when both source and destination are in the same row, i.e., partition; the same applies when both source and destination are in the same column, or in general, in a given subset of dimensions.

RVLB avoids this problem by limiting the selection of the intermediate switch to the local partition when the source and destination terminals belong to the same partition. If source and destination are in different partitions, RVLB behaves as the original VLB routing algorithm. The definition of a network *partition* is topology-dependent.

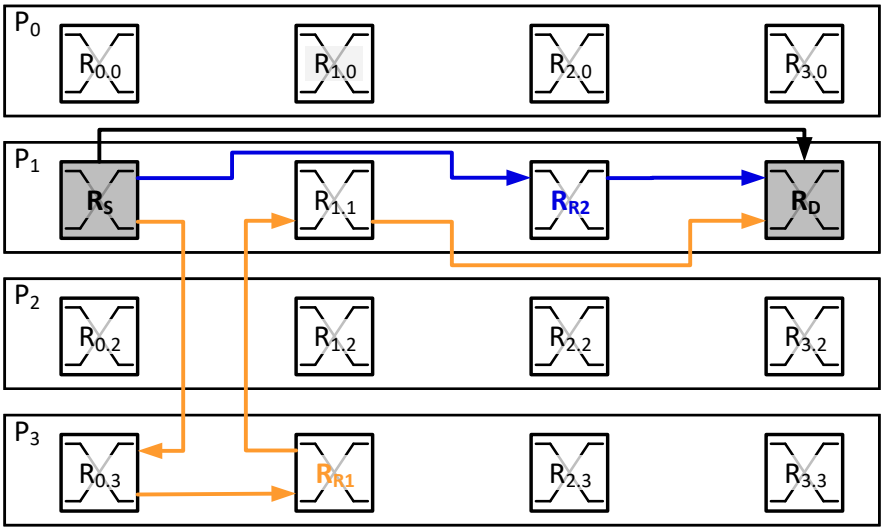
Considering a Dragonfly network, RVLB can be applied to packets with source and destination in the same partition, this is, it can be applied to intra-group traffic. In this case, RVLB only selects an intermediate switch from the local group, which shortens the non-minimal paths. Pathological congestion effects may occur with such traffic pattern which is denoted as *Adversarial Local* (ADVL) and is presented in Section 3.2.2.1.3. An example of this traffic pattern appears when all terminals attached to a router communicate with compute hosts in the next switch. In such case, original VLB avoids the congestion selecting an intermediate switch R_{ROOT} in a remote group, so the packet leaves the source group[2] to return to it later. By contrast, RVLB only selects an intermediate switch R_{ROOT} within the local group, which generates shorter paths and still avoids the congestion issue in the minimal link.

RVLB can also be applied to the selection of non-minimal paths in adaptive routings which are build upon Valiant load-balancing algorithm. When both source and destination terminals are in the same partition, the non-minimal path considered by these routings can be restricted, so the packet is first sent to an intermediate switch in the local partition and then sent minimally to its destination.

[2]Note that the source group is also the destination group.



(a) Dragonfly



(b) Flattened Butterfly

Figure 5-1: Example of MIN, VLB and RVLB between source (R_S) and destination (R_D) routers in DF and FB topologies. Note that some network links are omitted by simplicity. The intermediate switches are: R_{R1} and R_{R2}. In both cases, the Valiant path shows an example of turn-around problem without traversing the same switch twice. The *restricted* Valiant path is shorter than VLB and avoids pathological congestion within. MIN is obviously the shortest path in every case.

5.3 VLB-Recomp: Valiant Load-Balancing Recomputation

Valiant routing randomizes the paths of the packets to balance the use of network resources. However, transient congestion may appear, which generates *Head-of-Line* (HoL) blocking and delays injection.

This issue can be mitigated with *Valiant Load-Balancing Recomputation* (VLB-Recomp), also named Valiant with *recomputation*. In this mechanism, the selection of the intermediate switch R_{ROOT} for each packet is performed at the source router, as in the original mechanism. Whenever a packet is at the head of its injection buffer and the required output port, which is at the beginning of phase A, is blocked, the routing algorithm recomputes a new random intermediate switch in the routing computation stage. The detection of a blocked packet at the head of the buffer can be implemented in modern routers like the blocked counters [49] at arbiter inputs implemented in the Cray Aries ASIC [13], which increment their value if a flit is available at the input but instead arbitration has selected a different input. The recomputation of the intermediate router usually change the output port and ease packet injection, increasing throughput. Such recomputation may occur several times, until the packet is injected; once the packet is in-transit, the R_{ROOT} does not change. VLB-Recomp can be combined with the RVLB mechanism from Section 5.2, limiting the recomputation of the intermediate switch to the subset of allowed switches.

Unlike the original VLB, VLB-Recomp is not *oblivious*, because the intermediate destination is modified depending on the status of the network. According to the taxonomy detailed in Section 2.3, VLB-Recomp is *adaptive* and *congestion-oblivious*, because it routes based on the availability of output ports and not on an estimation of the network congestion.

The idea of recomputing the VLB intermediate switch can also be applied to adaptive routings, such as *Piggyback* (PB, [96]): when the required destination port for the selected intermediate switch is blocked, the switch's RC unit recomputes the R_{ROOT} and performs a new routing decision. With this modification, Piggyback with recomputation remains *adaptive* and *congestion-aware* as in the original PB implementation, because it routes based on the availability of output ports due to recomputation, but also based on an estimation of the network congestion to determine if a minimal or non-minimal path should be used.

5.4 ACOR: Adaptive Congestion-Oblivious Routing

This section proposes ACOR: *Adaptive Congestion-Oblivious Routing*. After a just following initial overview, its design is presented in Section 5.4.2 and its application to Piggyback adaptive routing is discussed in Section 5.4.3.

5.4.1 Motivation and overview

This section motivates the use of a routing which combines short and long paths for VLB phase A based on the network conditions, considering the trade-off between the length of the path and performance results under pathological traffic patterns.

As explained in Section 2.3.2, multiple *VLB phase A policies* can be used to generate the path for phase A which routes the packet from source to the intermediate router. They are

presented in Table 5-1. The three shorter options experience performance issues under pathological traffic patterns, as explained in Section 2.3.2. Usually, the trade-off has been solved using the longest path [15, 60], which avoids any pathological situation at the cost of the largest base network latency; The phase A precedes the minimal path *lgl* of phase B.

Table 5-1: Considered VLB phase A policies.

Name	Maximum phase A path length	Longest VLB path
VLB_{lgl}	3 hops	$l_1\, g_1\, l_2 - l_3\, g_2\, l_4$
VLB_{lg-}	2 hops	$l_1\, g_1\, - l_3\, g_2\, l_4$
VLB_{-gl}	2 hops	$g_1\, l_2 - l_3\, g_2\, l_4$
VLB_{-g-}	1 hop	$g_1\, - l_3\, g_2\, l_4$

The goal of ACOR is to optimize the common case providing minimal latency, while supporting pathological traffic patterns with longer paths. This is implemented by adapting the selected VLB phase A policy to the network conditions. The implementation of the VLB phase A policies in Table 5-1 is based on the restriction mechanism introduced in Section 5.2. Indeed, a VLB phase A policy can be seen as applying a restriction on the allowed intermediate switches under global traffic: VLB_{-g-} restricts the selection of the random intermediate destination to switches directly connected to the source switch, but belonging to different group, VLB_{-gl} restricts to any of the switches in groups directly connected to the source switch, VLB_{lg-} restricts to any of the switches directly connected to the source group[3] and VLB_{lgl} does not apply any restriction to the selection and whichever switch within the whole interconnection network can be selected.

5.4.2 ACOR design

ACOR employs a *path A policy sequence*, which is a sequence of VLB phase A policies, ordered from the shortest to the longest path. The number of policies in a given sequence is defined as the *sequence level*. Table 5-2 presents three sequences used in this chapter, with two levels (2L-A and 2L-B) or three levels (3L). The selection of these specific policies is discussed in Section 5.4.2.1.

The sequence used is fixed for a given implementation. ACOR switches from one policy to another in the sequence based on the *recomputation* technique introduced in Section 5.3. The *ACOR level*, or simply *level*, defines the currently selected phase A policy from the sequence. When a packet cannot be injected because the output port is blocked, its path is

[3]Note that the source group could be also selected as the intermediate destination; in such case, the hop (g) in phase A is omitted.

Table 5-2: ACOR path A policy sequences. Phase A policies are summarized in Table 5-1.

Name	Label	Sequence		
Two-levels (A)	2L-A	$VLB_{-g-} \longrightarrow VLB_{lgl}$		
Two-levels (B)	2L-B		$VLB_{-gl} \rightarrow VLB_{lgl}$	
Three-levels	3L	$VLB_{-g-} \rightarrow VLB_{-gl} \rightarrow VLB_{lgl}$		
Path Length:		1	2	3

recomputed. In such case, the ACOR level for the given packet can change towards longer paths before recomputing the path. When a routing is computed, the intermediate switch is selected according to the restrictions imposed by the current ACOR level, i.e., the current phase A policy.

An ACOR level can be maintained per individual packet or per switch. In the latter option, all the packets injected in each switch during a certain period of time are routed following the sequence imposed by the switch's ACOR level. These two alternative implementations, denoted as *ACOR-Packet* and *ACOR-Switch* respectively, are described in Section 5.4.2.2.

5.4.2.1 Selection of a VLB phase A policies sequence

Table 5-1 presents the available VLB phase A policies in the Dragonfly. Since there are policies with three different maximum path lengths, it makes sense to select sequences with two or three policies, in increasing order of path length. All the sequences need to end in the largest policy (VLB_{lgl}), which corresponds to the original VLB definition and avoids any pathological congestion introduced by shortening the path in phase A. Short paths are used first, to reduce base latency in absence of congestion.

Two different policies with length 2 are present in Table 5-1: VLB_{-gl} and VLB_{lg-}. Compared to VLB_{-g-}, each of these policies solve one different pathological congestion problem, as discussed in Section 2.3.2. The second local hop in VLB_{-gl} avoids the pathological congestion in the intermediate group under *Adversarial shift* traffic pattern with offset h (ADV+h) presented in Section 3.2.2.1.2, whereas the first local hop in VLB_{lg-} avoids the unfairness and certain congestion to a lesser extent under *Adversarial Consecutive* traffic pattern (ADVC) presented in Section 3.2.2.1.4.

The selection of which of these policies to use in a sequence is based on two arguments. First, it is relevant which of the two problems occurs at a lower load. Saturation caused by ADV+h occurs at load $\frac{1}{h}$ phits/node/cycle, for example 0.16 to 0.12 phits/node/cycle for $h \in \{6 - 8\}$. The unfairness and congestion under ADVC occur close to saturation, near 0.5 phits/node/cycle. These effects are observed in the evaluation in Section 5.5. Second, the problems under ADVC lie in the source group, while congestion under ADV+h occurs

in a remote group (the intermediate Valiant group, G_{ROOT}), which is more difficult to detect at injection time to make an accurate and early decision. For both reasons, it is reasonable to select VLB$_{-gl}$ over VLB$_{lg-}$, since it solves the congestion in the remote group that would otherwise appear at low loads.

With the selection of the length-2 phase A policy VLB$_{-gl}$, the three resulting sequences are presented in Table 5-2. Two sequences with two levels are considered: 2L-A presents the lowest base latency by starting with VLB$_{-g-}$ but switches to longer paths at lower loads; 2L-B gives an intermediate latency for a wider range of loads. The three-level policy 3L presents a more complex implementation, but tries to optimize a wide range of loads.

5.4.2.2 ACOR level management

The *ACOR level* of each packet at the head of an injection buffer indicates which phase A policy from the sequence is used when routing the packet. The *recomputation* mechanism is employed when packets cannot be injected, which is a sign of congestion issues created by the traffic pattern, the load and the current ACOR level. This mechanism is leveraged to raise the level towards longer paths when packets get blocked repeatedly. Each ACOR level is managed independently, but the implementation details differ when the management occurs per packet or switch.

In *ACOR-Packet* all packets start with the minimum level, so the shortest path is selected by default. Before the recomputation is performed, the ACOR level of each packet is raised when the output port is unavailable. When the ACOR level reaches the longest policy (VLB$_{lgl}$, Tables 5-1 and 5-2), it remains in such policy until the packet is injected. Since the amount of recomputation increases with the offered load and pathological congestion issues in the network, packets quickly adapt to use longer paths in presence of congestion.

In *ACOR-Switch* an *ACOR switch level* is derived from blocked output ports throughout the switch. All the packets injected in the switch during a certain period of time are routed following the sequence imposed by this switch level. The ACOR level of the switch is initialized with the first level of the sequence and increases and decreases according to the amount of blocked packets. Since this value adds information from blocked packets of many individual ports, it does not increase with every blocked packet. Instead, different thresholds are used, together with a hysteresis mechanism to provide stability and avoid oscillations.

For each transition in the path A policy sequence, ACOR-Switch employs two thresholds to increase (IT_1, IT_2) and decrease (DT_1, DT_2) the ACOR switch level. A blocked packet counter is maintained and a *Hysteresis Interval* (*HI*) is defined. The values for these parameters have been determined empirically and presented in Section 5.5.4.3. The blocked packet counter is reset at the end of every hysteresis interval. The switch level increases when the number of blocked packets exceeds the current increase threshold, without waiting for the end of the hysteresis interval. By contrast, the level is decreased only when the number of blocked packets is lower than the current decrease threshold at the end of the interval. This

design boosts a quick change to longer non-minimal paths when congestion is detected and a slow return to smaller non-minimal paths when congestion disappears.

Such design is sensible, since it continues injecting the packets through a longer VLB phase A policy up to the end of the interval, which is more than enough to avoid congestion by increasing slightly the average latency. However, not changing to a longer phase A policy on time introduces much more congestion.

5.4.3 PB-ACOR: adaptive Piggyback with ACOR

ACOR can be used as the basis of a non-minimal source-adaptive routing algorithms, such as UGAL or Piggyback. These routing algorithms select between minimal and non-minimal paths at injection based on an estimation of the network congestion. In an ACOR-based implementation, the non-minimal path is defined by the current ACOR level from the path A policy sequence.

In these routing algorithms, packets may be blocked when they want to advance following minimal or non-minimal paths. Packets that are blocked when trying to follow minimal paths are not considered for increasing the ACOR level nor the hysteresis mechanism in ACOR-Switch. For this reason, appropriate thresholds may differ from the original ACOR implementation without these underlying adaptive routing algorithms.

5.5 Evaluation

Firstly, this section presents the particular simulator configuration for the evaluations performed in this chapter and then, the performance results of the proposals. First, both introduced techniques, *RVLB* and *VLB-Recomp*, are evaluated in Sections 5.5.2 and 5.5.3. During the evaluation, the latter includes the former one because, as it will be seen later, RVLB performs equally or better than VLB. Next, the ACOR adaptive routing introduced in this chapter, which leverages the previous mentioned techniques to modify the length of non-minimal paths, is evaluated both stand-alone and combined with Piggyback respectively, in Sections 5.5.4 and 5.5.5.

5.5.1 Simulator configuration

The simulation experiments designed to evaluate the performance of the proposals introduced in this chapter in Sections 5.2, 5.3 and 5.4 have been carried out according to the methodology explained in Chapter 3. Details about the traffic pattern used in the evaluation experiments can be found in Section 3.2.2. The network simulator mimics the behavior of RVLB and VLB-Recomp techniques and ACOR and PB-ACOR routing algorithms, as described in the aforementioned sections.

Oblivious routing algorithms, *Minimal* (MIN) and *Valiant Load-Balancing* (VLB), implemented as explained in Sections 2.3.1 and 2.3.2 have been used because they provide the best performance under uniform or adversarial traffic patterns. Four phase A path lengths for VLB are considered: *lgl, lg-, -gl, -g-*.

Piggyback is included as an adaptive reference that implements per-packet source-adaptive regional congestion-aware routing algorithm, relying on state information for each global channel within a particular group. PB considers a global channel (*gc*) as saturated if the inequality shown in Equation 2-2 is satisfied. This information is distributed among switches of the group. PB employs *lgl* for phase A path and requires 4 and 2 virtual channels to avoid deadlocks on local and global links respectively.

The combination of the base simulation parameters presented in Table 3-1 and the particular ones for this chapter listed in Table 5-3 determine the network parameters and the configuration for ACOR and PB-ACOR routing algorithms, unless otherwise stated during a particular experiment.

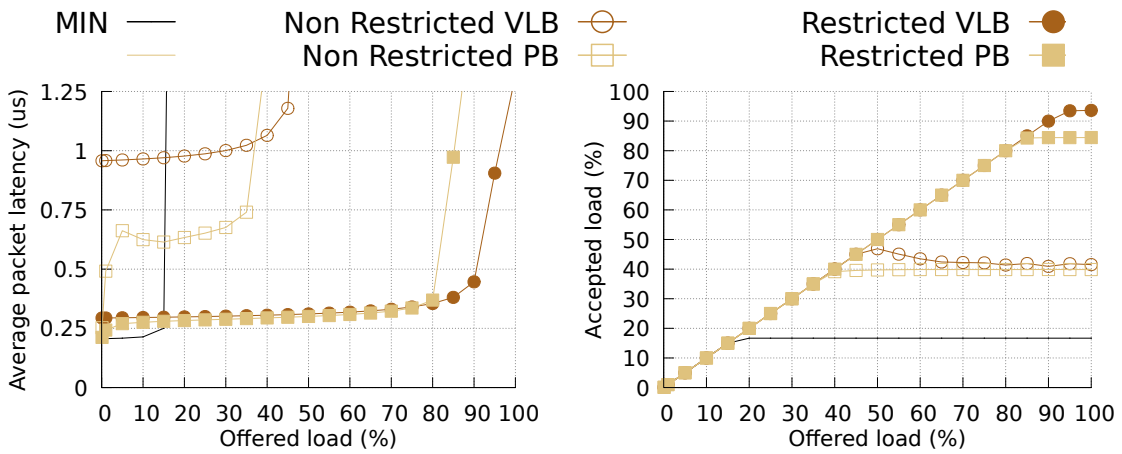
Table 5-3: Particular simulation parameters used to evaluate the proposals of this chapter.

	Parameter	Value
ACOR	Switch hysteresis interval	$HI = 500$ ns
	Increase level thresholds	$IT_1 = 15, IT_2 = 500$
	Decrease level thresholds	$DT_1 = 5, DT_2 = 15$
PB-ACOR	Switch hysteresis interval	$HI = 500$ ns
	ACOR increase level thresholds	$IT_1 = 15, IT_2 = 50$
	ACOR decrease level thresholds	$DT_1 = 5, DT_2 = 15$

5.5.2 RVLB performance results

Figure 5-2 presents the average packet latency and throughput results for *adversarial local* traffic pattern, explained in Section 3.2.2.1.3, using *minimal* routing algorithm as a reference and Valiant load-balancing and Piggyback routing algorithms with and without applying to them the *restricted* technique. The ADVL traffic pattern is used in this evaluation instead of *Adversarial shift* (ADV+*i*) because there is no intra-group traffic in, so VLB and RVLB behave exactly the same under such traffic. ADVL pattern solely presents intra-group traffic, so the impact of RVLB over VLB is significant, as can be seen in the presented figure.

Using MIN the local link connecting pairs of neighbor switches becomes a bottleneck, and only $\frac{1}{p} = 16.6\%$ of the traffic can be delivered using the minimal routes. Before this saturation point, MIN presents optimal latency. VLB and PB with VLB_{lgl} phase A policy raise



(a) Latency (b) Throughput

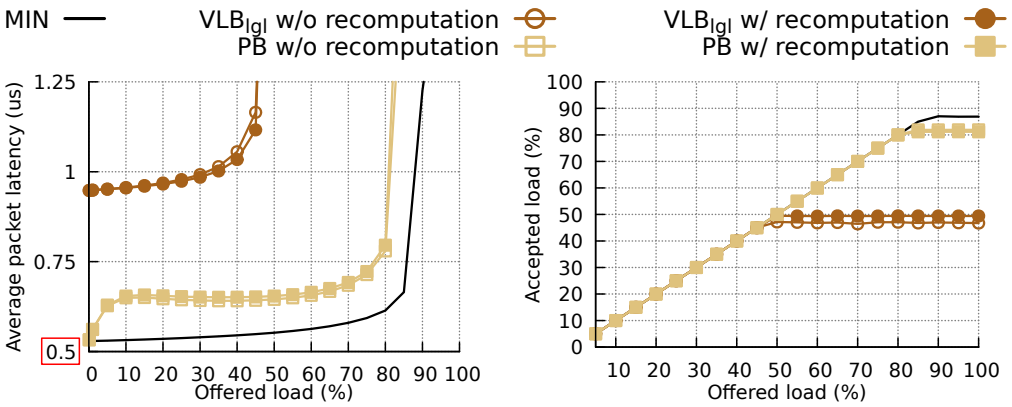
Figure 5-2: Average packet latency and throughput of VLB and PB using and not using the restricted technique under ADVL traffic pattern.

the saturation point of the accepted load near 50%. However, latency below the saturation point increases significantly, due to the extra cost of the two additional global hops to and from a remote intermediate group, and throughput is limited below 50%. Both VLB and PB using the restricted technique accept almost 100% of the offered load, since they avoid the turn-around problem and generate short paths as depicted in Figure 5-1, even though the throughput obtained by PB at high loads is lower due to the use of minimal paths. Furthermore, the latency is significantly lower than the one achieved by VLB, with a 69.9% reduction for a 30% load.

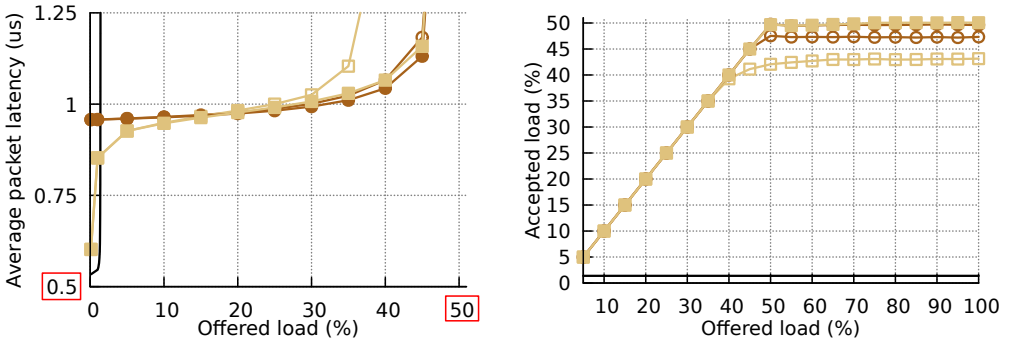
In conclusion, the restricted variants of VLB and PB performs equal or better than the original ones under the evaluated traffic pattern on this chapter or on previous work [21]. Hence, in upcoming sections all references to the mechanism are denoted simply as *VLB* and *PB* to refer to *RVLB* and *RPB* respectively.

5.5.3 VLB-Recomp performance results

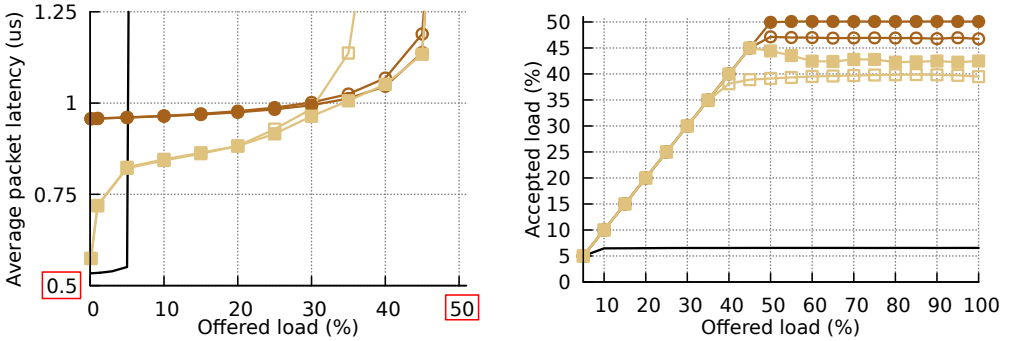
Figure 5-3 presents average packet latency and throughput results for different traffic patterns using VLB and PB (employing *restricted* technique) with and without *recomputation* technique (VLB-Recomp and PB-Recomp), as explained in Section 5.3. MIN is included as a reference. Latency results are presented in the load region before saturation and, in all cases latency is improved by recomputing the intermediate destination. This is expected, since the re-computation occurs when packets cannot be injected because of congestion in the originally selected path; the recomputation mechanism selects another path, and injects traffic earlier, so packets accumulate a lower latency. In the case of ADV traffic (which typically requires VLB routing algorithm) with a load of 40%, average latency is reduced by 11.4% when using recomputation.



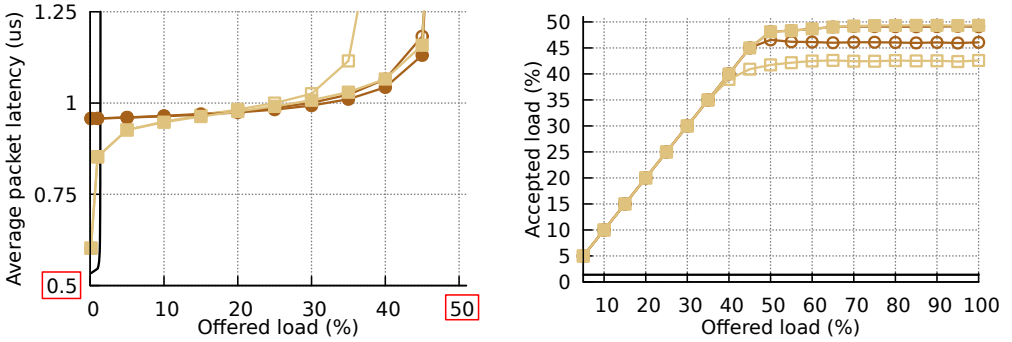
(a) Average packet latency and throughput under UN traffic.



(b) Average packet latency and throughput under ADV+1 traffic.



(c) Average packet latency and throughput under ADVC traffic.



(d) Average packet latency and throughput under ADV+*h* traffic.

Figure 5-3: Average packet latency and throughput of restricted Valiant load-balancing, with and without recomputation, under different traffic patterns.

Throughput results after the saturation point differ for each traffic pattern. Both UN and ADV+1 present a similar trend, with the original VLB routing algorithm presenting high variability and slightly reduced throughput. The reference with MIN routing is significantly different, with higher throughput under UN due to a lower usage of the global links, and very poor throughput under adversarial traffic because only one global link is used to carry all the traffic from every group, limiting the accepted load to $\frac{1}{a \cdot p} = \frac{1}{2h^2} = 1.38\%$. As mentioned above, Valiant with *recomputation* injects packets earlier, which leads to higher throughput, with an increase between 4.3% and 5.9%.

An evaluation of *restricted* Valiant with *recomputation* using multiple injection buffers and testing different allocation policies has been carried out in a previous work [21] and it is not included here for simplicity. In general, the use of multiple injection buffers is detrimental to the achieved throughput. However, using recomputation helps to alleviate the performance degradation when the congestion is not completely uniform, leveraging the less congested links to dispatch packets faster, and presents a lower overall impact due to use of multiple injection buffers than the RVLB without recomputation.

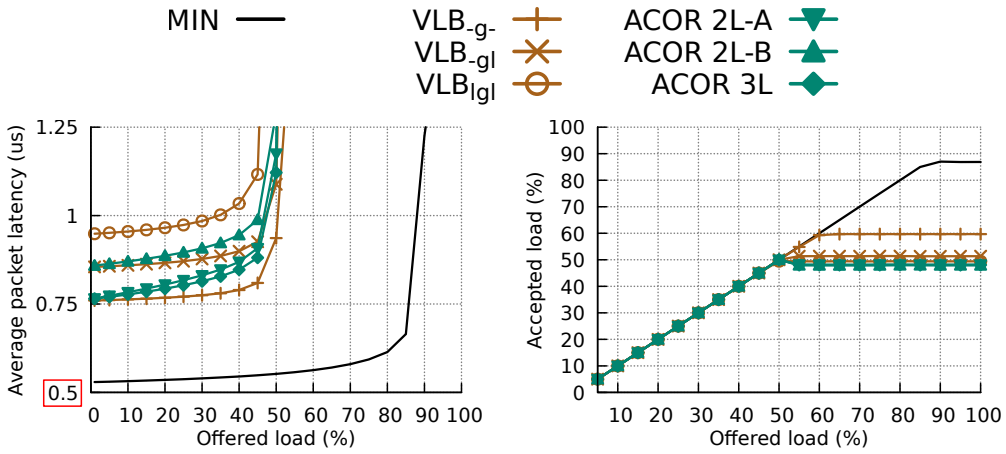
As VLB-Recomp and PB-Recomp perform equally or better than VLB and PB, respectively, for all the evaluated traffic patterns, the subsequent evaluations employ VLB-Recomp as VLB and PB-Recomp as PB. Therefore, the rest of this chapter employs *restricted* and *recomputation* techniques as a baseline.

5.5.4 ACOR performance results

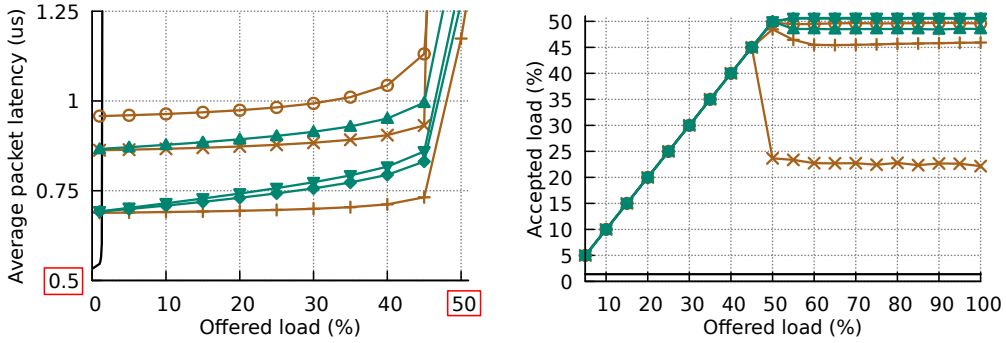
This section evaluates the performance of *ACOR-Packet* and *ACOR-Switch* under steady-state traffic in Section 5.5.4.1 and under transient traffic loads in Section 5.5.4.2. It also analyzes the selection of configuration parameters through a sensitive analysis in Section 5.5.4.3.

5.5.4.1 Performance under steady loads

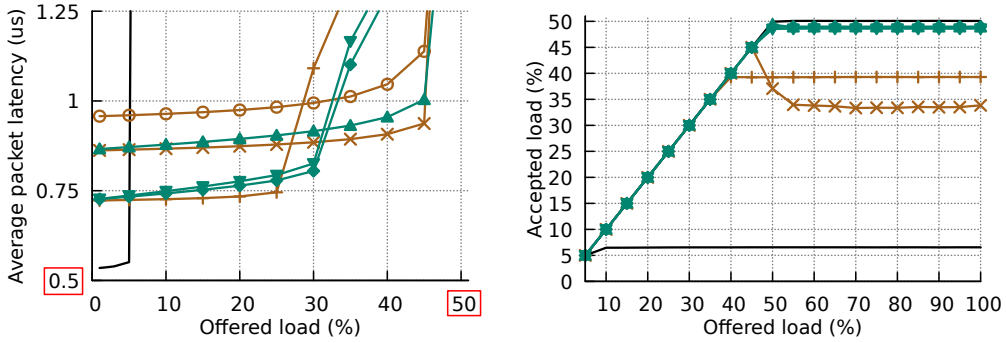
Figure 5-4 presents average packet latency and throughput of *ACOR-Packet* under UN, ADV+1, ADVC and ADV+*h* traffic patterns. As a reference, the result with MIN and VLB routing is also displayed; MIN is the baseline reference under UN traffic since it achieves minimal latency. Results with VLB routing consider the three relevant VLB phase A policies in Table 5-1: VLB_{-g-}, VLB_{-gl} and VLB_{lgl}. Each of these policies has a higher base latency over the previous ones, due to the additional local hops introduced. On the other hand, VLB_{lgl} presents the highest throughput under adversarial traffic patterns, since its better randomization avoids pathological congestion. VLB_{-gl} achieves lower latency at medium loads under ADVC and ADV+*h* traffic. This confirms the analysis in Section 5.4.2.1 where an optimal sequence of phase A policies employs VLB_{-g-} at low traffic loads and VLB_{-gl} at medium traffic loads to reduce latency, and VLB_{lgl} after the saturation point of VLB_{-gl} to achieve competitive throughput.



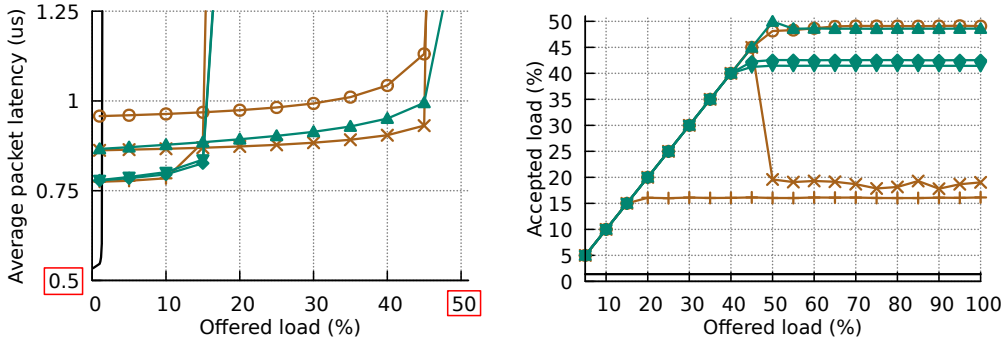
(a) Average packet latency and throughput under UN traffic.



(b) Average packet latency and throughput under ADV+1 traffic.



(c) Average packet latency and throughput under ADVC traffic.



(d) Average packet latency and throughput under ADV+*h* traffic.

Figure 5-4: Average packet latency and throughput of ACOR-Packet under different traffic patterns.

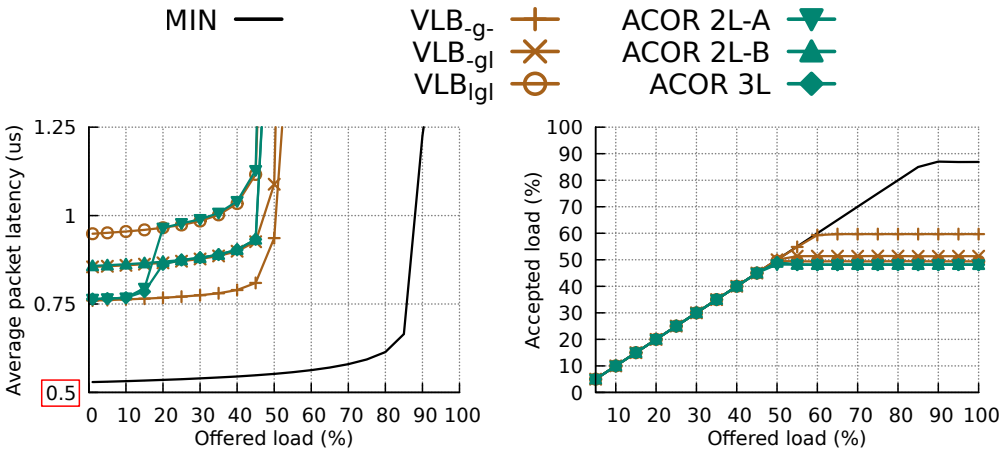
Figure 5-4 presents results for ACOR with the three *path A policy sequences* listed in Table 5-2. Since all of them employ the VLB_{lgl} phase A policy at the highest level, they achieve competitive throughput under all traffic patterns; under ADV+*h* there is a difference in throughput between policies, with sequence 2L-B outperforming the others. This occurs because under this traffic there is a pathological effect of congestion in the intermediate group which requires a non-minimal local hop, as discussed in Section 2.3.2. However, this effect occurs in a remote group, and ACOR relies on the congestion spreading back to the source group in order to trigger a change in the ACOR level. The 2L-B sequence is able to reach higher throughput because all the levels use a VLB phase A policy that selects an intermediate switch instead of a group; for the same reason, the other policies suffer from high latency under medium loads.

The 2L-A and 3L sequences present similar latency curves, albeit slightly lower for the 3L mechanism. This occurs because the 3L sequence has a lower proportion of packets using the highest level in the sequence, which is VLB_{lgl} policy, since two level changes are required to reach it. Therefore, the 3L variant is not effective in the per-packet implementation. The 2L-B sequence has a higher base latency due to the extra local hop at the source group, but lower latency at intermediate loads under ADVC and ADV+*h* traffic, because the additional local hop allows to mitigate congestion.

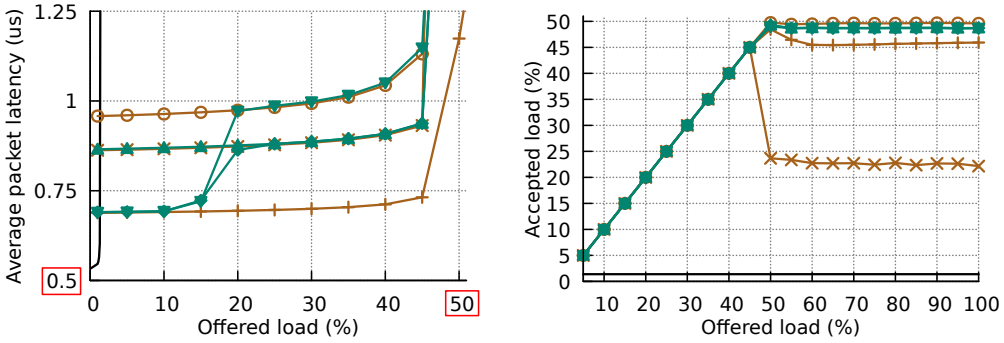
Figure 5-5 displays the performance results with *ACOR-Switch* under the same traffic patterns evaluated with ACOR-Packet.[4] This mechanism, although more complex than ACOR-Packet, is able to better identify the pathological congestion issues, using the same ACOR level for all the packets in the same switch. Throughput results are similar to those from ACOR-Packet except under ADV+*h* traffic, where all the sequences now achieve competitive performance. Managing the ACOR levels per-switch renders very different latency values, with the curves matching those from VLB with the different policies and rapid transitions between phase A policies. Latency with the 2L-B and 3L sequences is slightly higher than ACOR-Packet at intermediate loads, but stays relatively flat before the saturation point. This is particularly noticeable with the 3L sequence, which now fully transitions between levels to try to adapt to the optimal decision under all loads.

Another feature of ACOR is that it is able to adapt to different needs at different switches in the group. Figure 5-6 shows the average accepted load for all the hosts at each switch of the G_0 of the network, under a 0.5 phits/node/cycle load of ADVC traffic. Under this pattern, terminals at the last switch of the group have uneven access to the minimal global links, since they are all directly connected. This favors a higher amount of minimally-routed traffic at the bottleneck switch, but prevents them from sending traffic non-minimally when the VLB_{-g-} and VLB_{-gl} policies are used. ACOR exhibits a similar accepted traffic load for all the switches in the group with all the sequences, even 2L-A and 3L which employ VLB_{-g-}

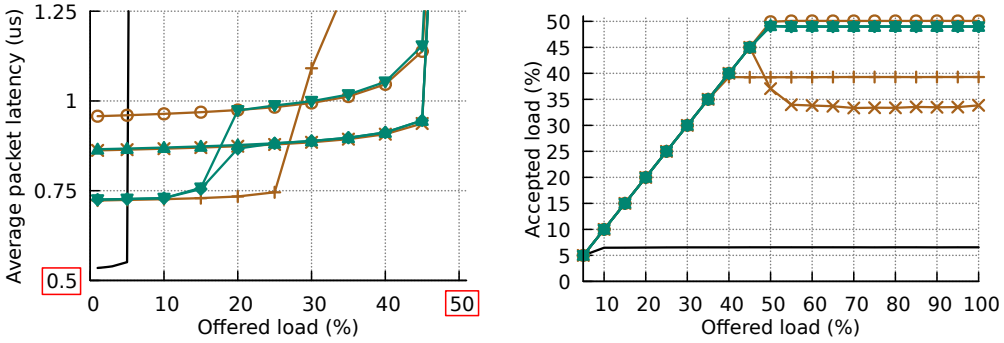
[4] Alternative traffic patterns, which are not adversarial but impose significant network congestion, were evaluated in [21] and are omitted here for concision and to keep the discourse in line with the whole dissertation.



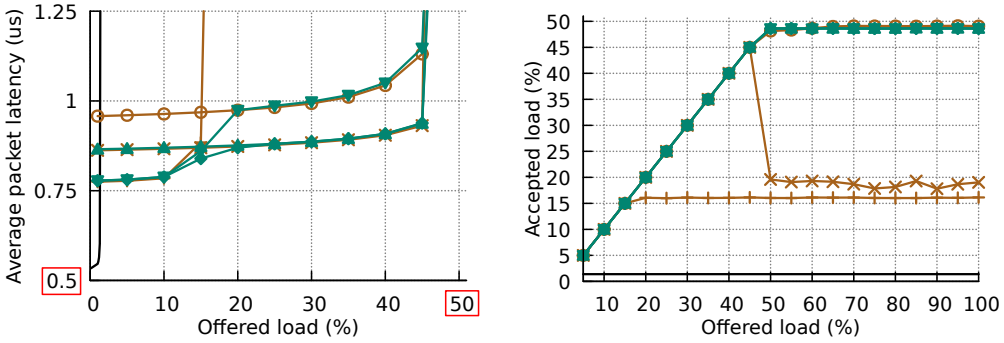
(a) Average packet latency and throughput under UN traffic.



(b) Average packet latency and throughput under ADV+1 traffic.



(c) Average packet latency and throughput under ADVC traffic.



(d) Average packet latency and throughput under ADV+*h* traffic.

Figure 5-5: Average packet latency and throughput of ACOR-Switch under different traffic patterns.

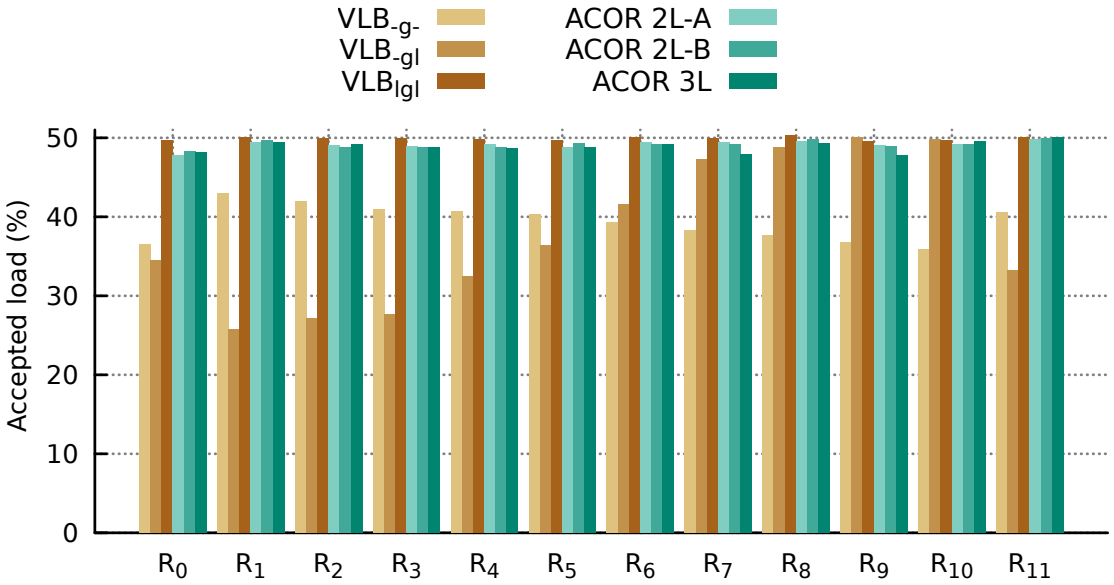


Figure 5-6: Averaged throughput accepted for each switch of G_0 under ADVC traffic pattern with a load of 50% using ACOR-Switch.

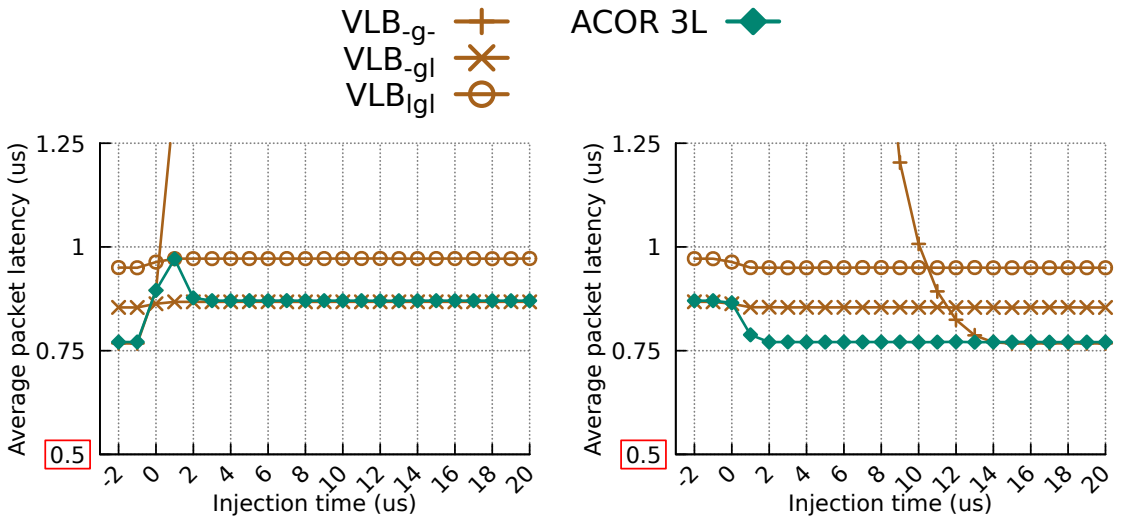
at the earliest level; the capability of switching between policies for each switch allows it to prevent pathological throughput unfairness effects as those suffered with VLB routing.

5.5.4.2 Performance under transient loads

A key benefit of ACOR is its ability to adapt to the network needs under different traffic loads, by changing the VLB phase A policy used. Figure 5-7 illustrates the effect of the transition between ACOR levels, displaying the average packet latency under ADV+*h* traffic that changes the traffic load. Results with *VLB* are provided as a reference. *PB* is not included as a reference because it always employs VLB_{lgl} for the packets routed non-minimally, so there is no change in its behavior in this experiment.

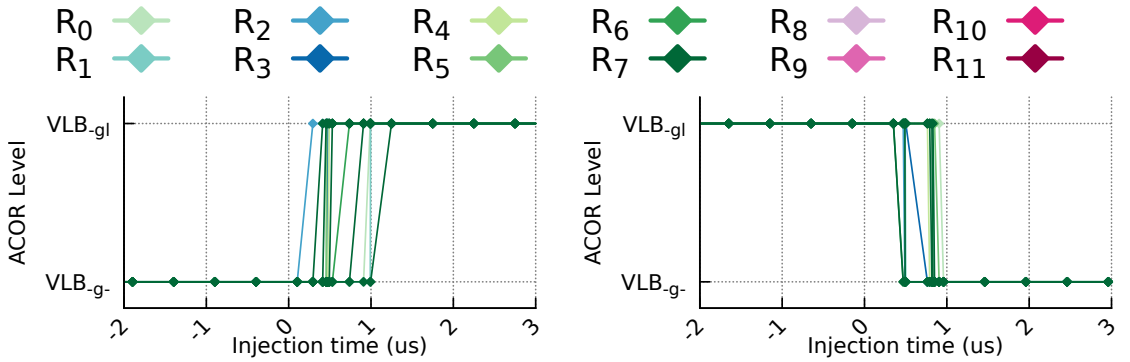
When the traffic load increases, in Figure 5-7(a), the VLB_{-g-} policy is no longer competitive and the latency of VLB with this policy increases significantly. ACOR transitions from using the VLB_{-g-} policy that provides the lowest base latency at low load (5%) to a VLB_{-gl} policy which has the lowest latency at high load (25%). Conversely, when the traffic load decreases, in Figure 5-7(b), ACOR changes from the VLB_{-gl} policy to VLB_{-g-}. In both cases, ACOR matches the latency of VLB with the most suitable phase A policy for each load, and is able to reach a steady behavior in less than 2 μs in both transitions. Results to transitory changes with other parameters resulting in other cycle durations are discussed later.

Figure 5-8 displays the current ACOR level for different switches of the same group, under the same transient traffic as in Figure 5-9. Certain switches change earlier to a higher level, depending on the amount of time it takes for the congestion to propagate back to them. In general, it can be observed that all the switches transition to a higher level in less than 1 μs, and to a lower level in less than 0.5 μs.



(a) From 5% to 25%. (b) From 25% to 5%.

Figure 5-7: Average packet latency of ACOR-Switch 3L under ADV+*h* traffic with transient traffic loads. At *t=0*, load transitions from (a) 5% to 25% and from (b) 25% to 5%.



(a) From 5% to 25%. (b) From 25% to 5%.

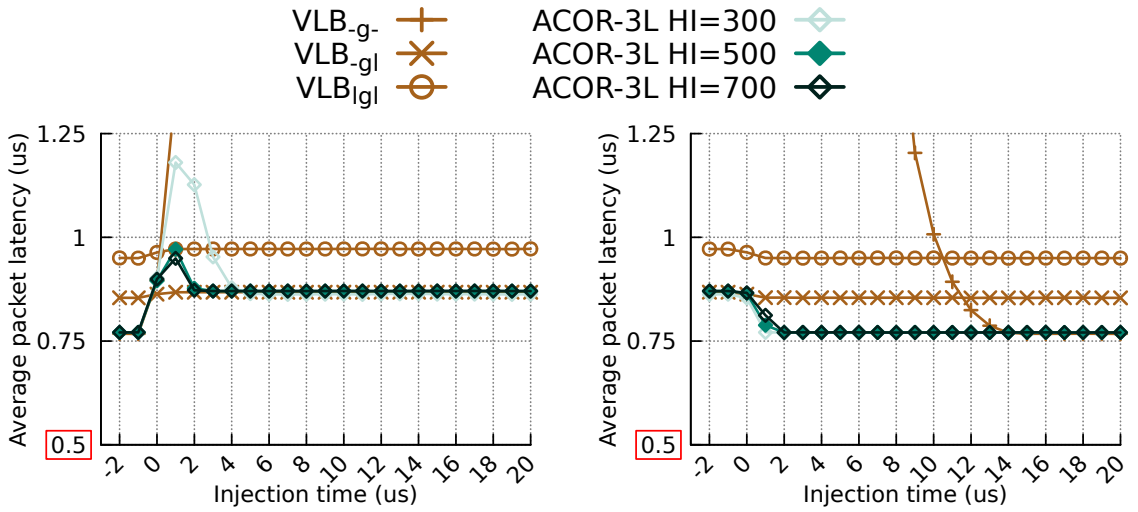
Figure 5-8: Evolution of the ACOR level of individual routers in a group using ACOR-Switch 3L under ADV+*h* traffic pattern with transient traffic loads. At *t=0*, load transitions from (a) 5% to 25% and from (b) 25% to 5%. Note that VLB_{lgl} level is omitted for clarity because it is not used.

5.5.4.3 Sensitivity analysis

ACOR-Switch relies on several parameters to manage the ACOR level of the switch. The ACOR level controls what phase A policy is used from those available in the sequence. The following paragraphs analyze each parameter, either alone or in groups, to present the robustness and the stability of the proposal.

5.5.4.3.1 Hysteresis interval: *HI*. The hysteresis interval defined in Section 5.4.2.2 determines the amount of time that ACOR waits before deciding to decrease a level, if the number of packets is below the threshold. Figure 5-9 presents different cycle durations to evaluate its impact in the sensitivity of ACOR-Switch with the three-level sequence. Results of different

VLB phase A policies are provided as a reference. The behavior in Figure 5-9(a) shows that when short hysteresis intervals are used, longer transition time with higher peak latency occurs, which may seem counter-intuitive. A short interval resets the blocked packet statistics more often, which delays the transition to a higher level in the sequence until the network congestion becomes more severe. By contrast, a long interval is slightly detrimental when the traffic load decreases because it delays the transition to a lower level.



(a) From 5% to 25%.

(b) From 5% to 25%.

Figure 5-9: Average packet latency of ACOR-Switch 3L under ADV+*h* traffic pattern with transient traffic loads to evaluate three different cycle durations. At *t=0*, load transitions (a) from 5% to 25% and (b) from 25% to 5%.

5.5.4.3.2 ACOR level management thresholds: IT_1, IT_2, DT_1 and DT_2. Two threshold values determine the decision to increase or decrease a level in the sequence, one to increase and one to decrease for every level transition. Figure 5-10 displays the impact of the increase threshold values with the three-level sequence, for a sweep in the traffic load under ADV+i traffic with $i \in \{1, h\}$. The first threshold controls the transition from $\text{VLB}_{\text{-g-}}$ to $\text{VLB}_{\text{-gl}}$, and the second threshold determines the change from $\text{VLB}_{\text{-gl}}$ to VLB_{lgl}. Results of VLB are used as a reference. Lower increase threshold values make routing more eager to transit to a higher level, increasing the latency, whereas higher thresholds force the routing to stay in the current level for higher loads, which can lead to network congestion. The latter effect is observed in Figure 5-10 with $IT_1 = 25$. The selected values $IT_1 = 15$ and $IT_2 = 500$ as the default parameters to evaluate the proposal represent a trade-off between both effects and achieve competitive performance under both patterns.

Similarly, Figure 5-11 shows the behavior for different decrease thresholds. In this case, the behavior is the opposite of the increase threshold: a lower value allows ACOR to transit more easily to a lower level. Since the transition to the highest level in the sequence occurs under high traffic loads, the impact of the second threshold is rather limited. However,

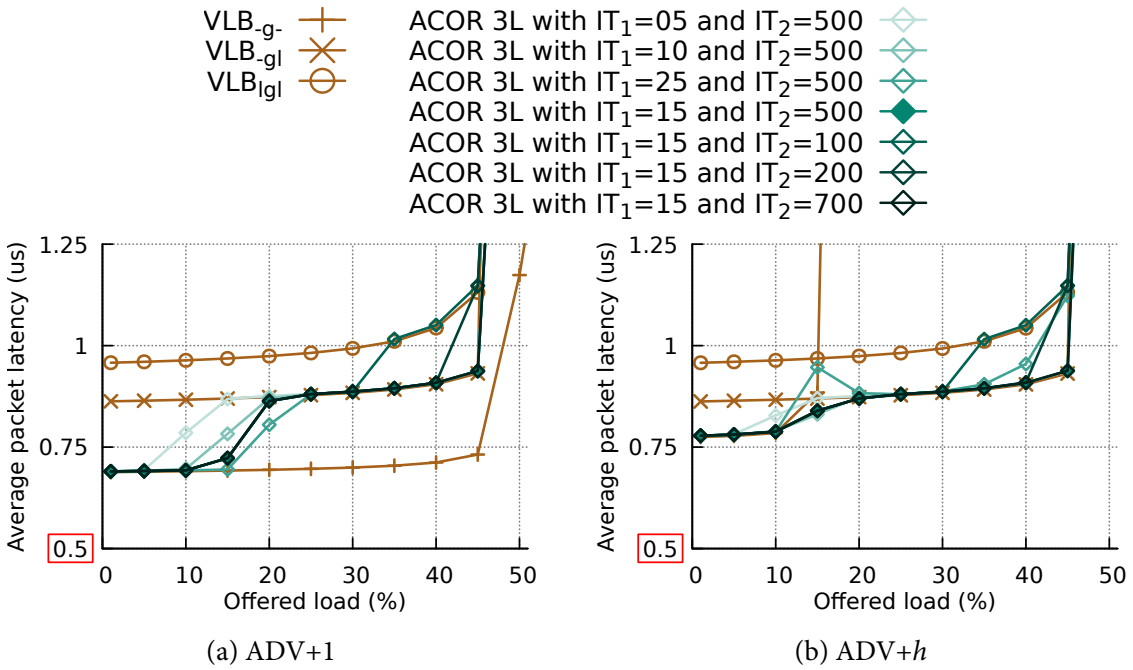
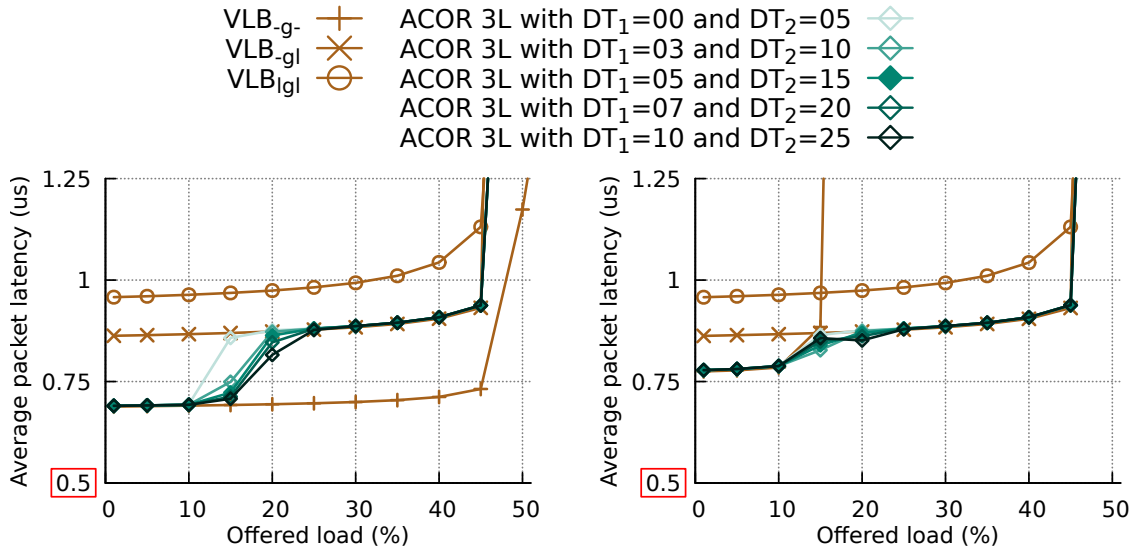


Figure 5-10: Latency of ACOR-Switch 3L with different increase thresholds (IT_1, IT_2) values under (a) ADV+1 and (b) ADV+*h* traffic patterns.

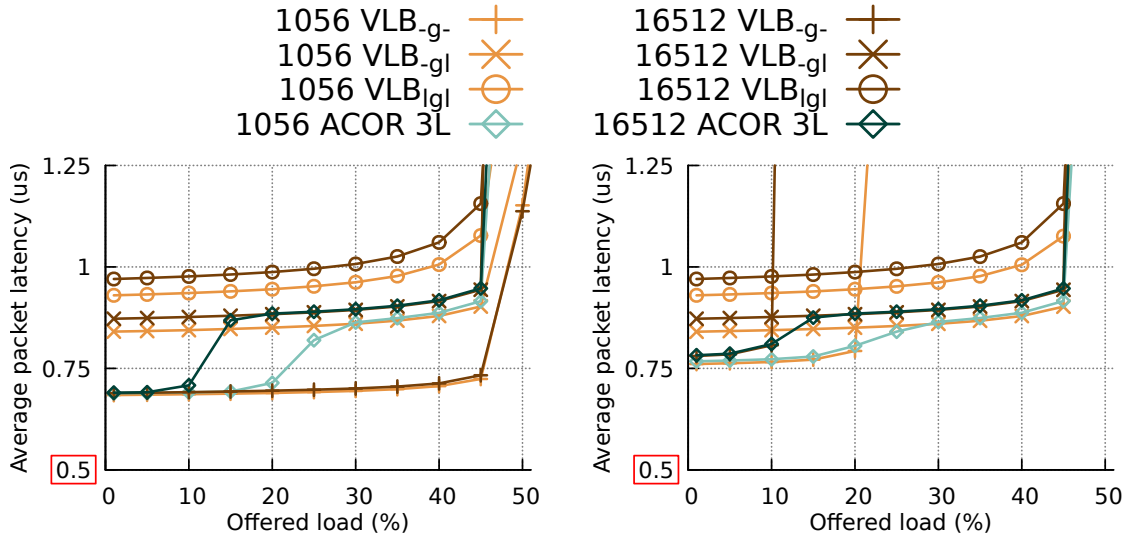
the first threshold has a significant impact in the latency at medium loads under ADV+1 traffic, making the routing algorithm more prone to higher latency with a lower threshold: lower values allow the routing to transition more easily to the previous level and provoke oscillations in the selection of the phase A policy. Again, the values $DT_1 = 5$ and $DT_2 = 15$ selected for the evaluation of the proposal represent a good trade-off in the performance results.

5.5.4.3.3 Network size. Figure 5-12 shows the behavior of ACOR-3L with a network size of 1,056 and 16,512 compute hosts; note that these curves can be compared with the results presented in Figure 5-5 for a network size of 5,256 compute nodes used across this dissertation. ACOR changes from VLB_{-g-} to VLB_{-gl} phase A policy when the latency employing VLB_{-g-} is triggered for each size under ADV+*h* traffic pattern and similarly jumps from VLB_{-gl} to VLB_{lgl} when VLB using VLB_{-gl} is saturated. The configuration parameters of three-level ACOR routing algorithm are different for each network size because the saturation point of each phase A policy under different traffic patterns is tied to the *h* parameter of Dragonfly topology. So, for the network size used across this work, the simulation parameters are presented in Table 5-3 and $IT_1 = \{20, 10\}$ and $DT_1 = \{8, 3\}$ are used for network sizes of 1,056 and 16,512 compute hosts respectively. As it can be concluded, the behavior of three-level ACOR in the simulation results is the same for different network sizes.



(a) ADV+1 (b) ADV+*h*

Figure 5-11: Latency of ACOR-Switch 3L with different decrease thresholds (DT_1, DT_2) values under () ADV+1 and (a) ADV+*h* traffic patterns.



(a) ADV+1 traffic. (b) ADV+*h* traffic.

Figure 5-12: Latency of ACOR-Switch 3L with two different network sizes (1,056 and 16,512 compute hosts) under (a) ADV+1 and (b) ADV+*h* traffic patterns.

5.5.5 PB-ACOR performance results

This section evaluates the performance of *PB-ACOR*, the variant of Piggyback presented in Section 5.4.3 that relies on ACOR for determining the non-minimal path length. Section 5.5.5.1 presents its performance results under steady-state traffic patterns and its performance under transient traffic loads is discussed in Section 5.5.5.2.

5.5.5.1 Performance under steady loads

Figure 5-13 presents results for PB-ACOR with three different path A policy sequences under the usual traffic patterns,[5] based on the ACOR-Switch variant. Parameter tuning for PB-ACOR has been carried out similarly to the ACOR case presented in Section 5.5.4.3. The parameters used to configure the network simulator are presented in Table 5-3, and differ from the parameters used in ACOR because the non-minimal path is not used for all packets, so the amount of blocking differs. The thresholds used in PB-ACOR make the change to VLB_{lgl} non-minimal paths *easier* than in ACOR. Since VLB_{lgl} gives better latency at intermediate loads, the latency of PB-ACOR using sequences 2L-B and 3L is more competitive than 2L-A. After saturation, all the adaptive mechanisms present a similar throughput.

Under UN traffic, the latency of the baseline PB is higher than MIN. This is explained by the system sending part of the traffic non-minimally, possibly caused by transient congestion [192]. Note that the models used in this thesis do not employ the history window proposed in [192] to mitigate transient congestion. By contrast, the latency of PB-ACOR is almost optimal (similar to MIN), up to a load of 75% using sequences 2L-B and L3. Apparently, the use of shorter paths provided by ACOR helps reduce the effect of transient congestion.

The effect in adversarial traffic is similar to ACOR. At low loads, the hop count is reduced and latency is improved. In the three adversarial patterns presented, latency at 10% load is reduced by 16.5% to 25.5%. At intermediate loads, both mechanisms start to behave similarly because both rely on VLB_{lgl}, and saturation throughput is equal.

5.5.5.2 Performance under transient loads

Same as ACOR, PB-ACOR is able to adapt the routing to the live network situation by changing the phase A policy. Furthermore, it delegates the selection between minimal or non-minimal paths to the underlying Piggyback. Figure 5-14 depicts the effect of the transition between different traffic patterns, displaying the average packet latency at the same offered load on both traffic patterns. Note that the offered load should be set before the saturation point, under both traffic patterns, because the experiment analyzes average packet latency. The percentage of misrouted packets has been evaluated as in Section 4.7.4.2 but is not presented for concision. Results of *VLB* and *ACOR* are provided as references that do not use minimal routing. MIN routing is not presented because the performed evaluations show that its reacting time is higher than 20 μs. *PB* adaptive reference reacts to the network congestion and selects between a minimal or non-minimal path for each packet.

When the traffic pattern changes from UN to ADV+*h*, in Figure 5-14(a), VLB_{-g-} saturates and its latency tends to infinity. The other two VLB curves remain stable because both sup-

[5] Alternative traffic patterns, which are not adversarial but impose significant network congestion, were evaluated in [22] and are omitted here for concision and to keep the discourse in line with the whole dissertation.

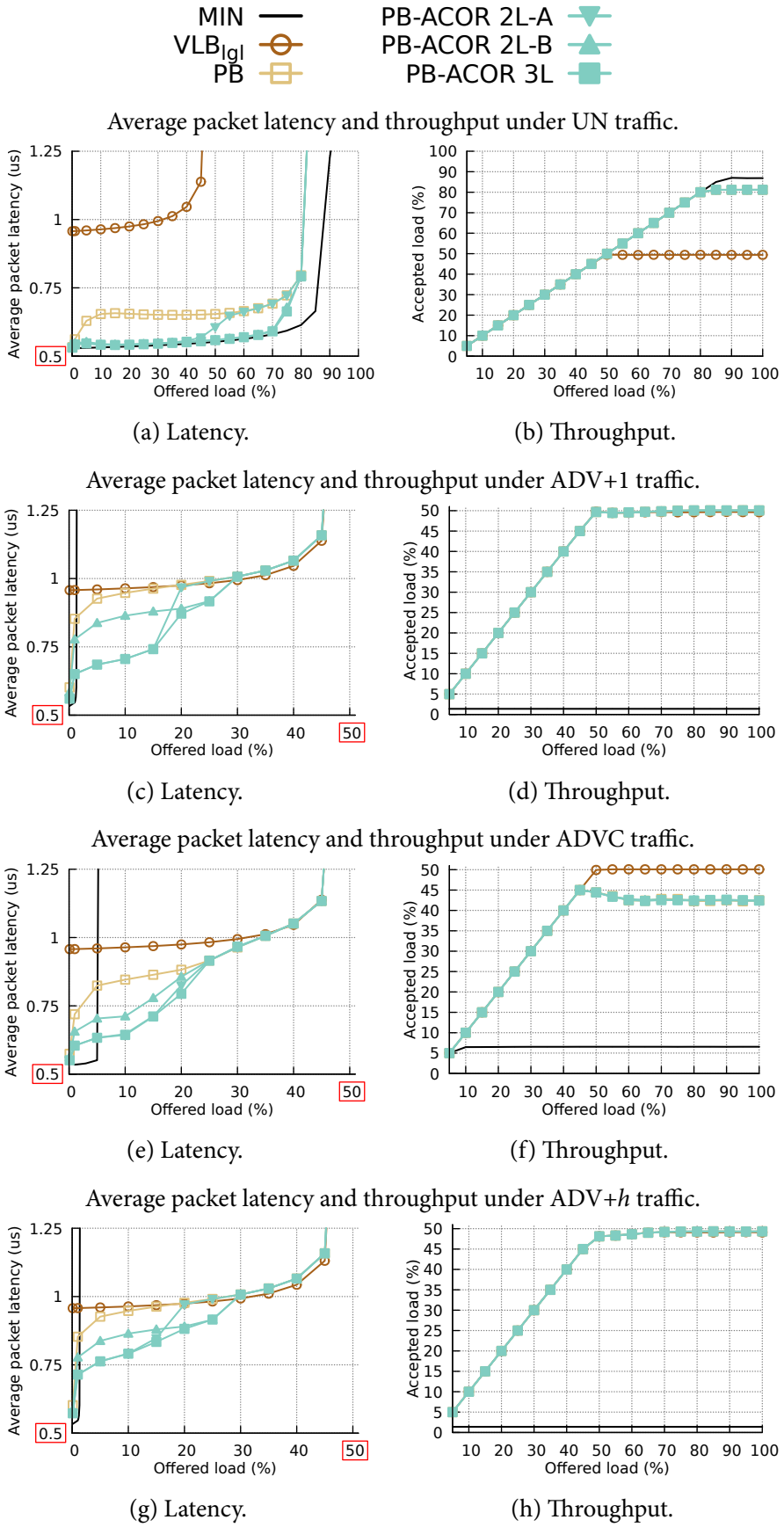
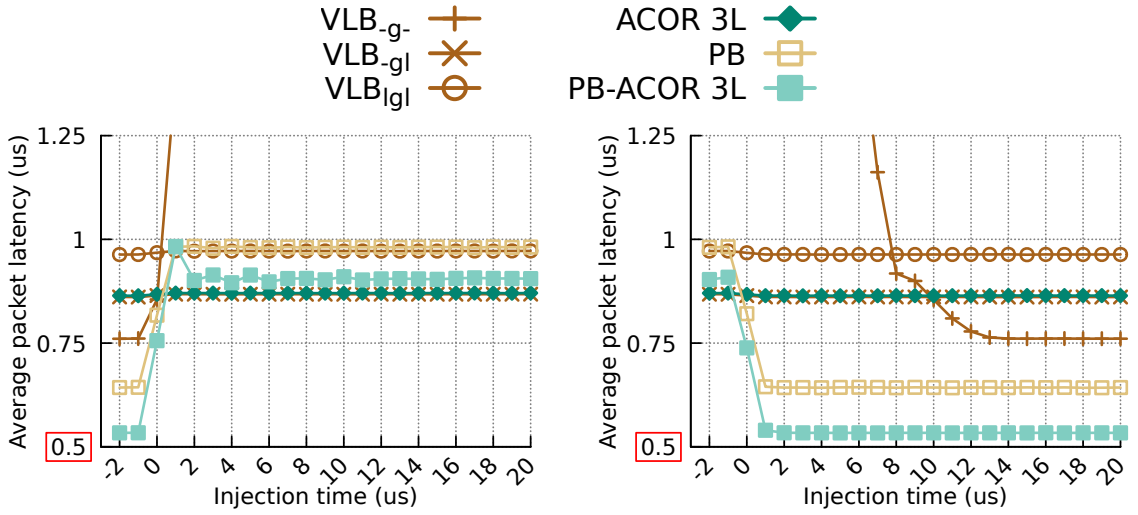


Figure 5-13: Average packet latency and throughput of PB-ACOR under different traffic patterns. MIN and VLB are presented as references.



(a) From UN to ADV+*h*.

(b) From ADV+*h* to UN.

Figure 5-14: Average packet latency of PB-ACOR 3L under transient traffic loads with an offered load of 25%. At *t=0*, load transitions from (a) UN to ADV+*h* and (b) ADV+*h* to UN.

port the ADV+*h* traffic pattern. ACOR also remains stable and its latency resembles VLB_{-gl}. PB's latency grows quickly, up to the same value obtained by VLB_{lgl}, because it increases the amount of non-minimally routed packets to adapt to the ADV+1 traffic. The average packet latency of PB-ACOR before $t = 0$ is the lowest because it resembles MIN, as it is presented in Figure 5-13(a). Beyond the traffic change, PB-ACOR starts to send packets through non-minimal paths and so it increases its average packet latency. As it can be seen, the adaptability of PB-ACOR to the traffic pattern is as quick as PB. However, after an initial step, PB-ACOR reduces its ACOR level and so, the obtained latency result. Conversely, in Figure 5-14(b), VLB_{-g-} recovers a non-saturated situation when the traffic changes from ADV+*h* to UN with a delay higher than 8 μs. As previously, the other two VLB curves and ACOR continue without any change. PB decreases the amount of traffic injected non-minimally and reduces its latency. Moreover, PB-ACOR quickly reduces the phase A path length for the packets sent non-minimally and obtains a slightly lower average packet latency than PB. In both cases, PB-ACOR employs the most suitable phase A policy for each situation, and it is able to reach to a traffic change from a steady situation in less than 2 μs in both transitions.

5.6 Conclusion

This chapter firstly introduces two improvements to Valiant routing, targeting high-radix networks: 1) *restricted* technique improves the performance of VLB for traffics with locality, selecting the intermediate router in the same network partition as the source and destination. Examples of such partitions have been introduced for Dragonfly and Flattened Butterfly networks; and 2) Valiant with *recomputation* avoids HoL blocking at injection by selecting

an alternative random intermediate router when the output port is stalled. Secondly, based on these two techniques and extending the restrictions in the intermediate path to global traffic, *ACOR: Adaptive Congestion-Oblivious Routing* is introduced.

The goal of ACOR is to optimize the common case providing minimal latency while supporting pathological traffic patterns with longer paths. It expands the idea of path recomputation to adapt the routing to network conditions, changing the phase A policy, which determine the path used for the non-minimally sent packets, following a given sequence ordered by path length. It prevents variability in the results through a simple hysteresis mechanism. ACOR needs, for each topology, the definition of the *partition* concept, which determines the elements that are considered as local, the available different length VLB phase A policies and relevant path A policy sequences. This chapter presents examples of some details of the design for folded-Clos, Flattened Butterfly and Dragonfly topologies. However, all the design details and the evaluation are explained over a Dragonfly topology. The implementation is relatively simple, according to the description presented in Section 5.4.2, and it can be extrapolated to other topologies. Three sequences of VLB phase A policies with different length have been considered in this chapter. Same as Valiant, ACOR does not send traffic minimally, so its performance under benign traffic is suboptimal. The ACOR mechanism has been coupled with a non-minimal adaptive routing, PB in the case of a Dragonfly network. The result of that combination is PB-ACOR, which selects the shortest feasible non-minimal path but only when the minimal route is congested. This mechanism maintains the benefits from ACOR for adversarial traffic and is competitive under uniform traffic.

Evaluation results show that all the ACOR variants avoid any throughput pathologies and reduce base latency by up to 28%, compared to a Valiant which has applied the *restricted* and *recomputation* techniques. The lowest latency values are achieved with a three-level sequence that exploits three different path A policies; however, the two-level sequence 2L-B presents similar results except below 15% loads, where its base latency is higher. ACOR also avoids pathological unfairness under ADVC traffic and react to the traffic changes in a steady situation in less than 2 μs. Two management strategies for the transitions in the sequence have been considered, per-packet and per-switch. Per-switch management achieves better performance as it considers the amount of blocked packets across the whole switch; however, per-packet management with the 2L-B sequence has similar performance except for higher latency at low loads, and lower implementation costs. PB-ACOR mechanism maintains the benefits from ACOR for adversarial traffic and is competitive under uniform traffic. It reaches high throughput and optimal latency under UN traffic, significantly outperforming Valiant, and improves base latency up to 25.5%. It also achieves rivaling throughput and a significantly lower latency compared to base PB. Moreover, its reaction time to traffic changes is as quick as PB to be in a non-saturated situation and less than 2 μs to achieve a stable situation. For these reasons, PB-ACOR with the three-level path A policy sequence and per-switch level management results outperforms the rest of evaluated mechanisms.

Latency-optimized Non-minimal Adaptive Routing for Dragonfly Networks

6

Low-diameter network topologies require non-minimal routing algorithms to avoid network congestion, such as *Valiant Load-Balancing* (VLB). However, it doubles path length and base latency. As discussed previously, while using shorter non-minimal paths may improve performance, it may also introduce congestion depending on the traffic pattern.

A main objective is that the routing algorithm determines at injection a path for the packets which introduces the minimum amount of additional hops while avoids network congestion issues. Previous chapter introduces a routing that pursues the same objective. However, that proposal does not maximize the performance in all situations nor takes into account the traffic pattern present in the network. One option to overcome the latter and observe what is happening in the network is to analyze the performance counters exposed by modern routers. An analysis of the impact of different *VLB phase A policies* and the idea of using performance counters to infer the traffic pattern present in the network are presented in Section 6.1.

This chapter proposes *LIAN: Latency-Improved Adaptive Non-minimal routing algorithm for Dragonfly networks* in Section 6.2. LIAN extends the use of traffic counters already present in modern routers to adapt non-minimal path length based on the inferred traffic pattern. It also modulates the routing decision based on live network conditions. Same as ACOR routing algorithm introduced in Chapter 5, the decision if route the packets minimally or non-minimally can be done by adaptive routing algorithms such as *Universal Globally Adaptive Load-balancing* (UGAL, [172]) or *Piggyback* (PB, [96]) or by the QCN-Switch architecture introduced in Chapter 4.

Evaluation results, discussed in Section 6.3, conclude that LIAN obtains almost-optimal latency and outperforms state-of-the-art adaptive routing mechanisms, reducing latency by up to 30.0%, providing stable throughput and throughput fairness. The path length is well adapted to the network conditions so, the latency is as low as possible in all evaluated scenarios. Depending on the traffic pattern detected, the path length is increased to avoid network congestion. This also is evaluated under transient traffic pattern to study the LIAN reaction time and performance results, exhibiting an almost-immediate reaction time. Additionally, the modulation of underline UGAL mechanism is very effective. Moreover, the evaluations shows that LIAN does not present routing unfairness.

The conclusions of this chapter, presented in Section 6.4, allow to classify LIAN as a *non-minimal source-adaptive local congestion-aware* routing algorithm that combines

the following desirable properties:

- relying solely on local information;
- employing the shortest VLB phase A path allowed by the live network situation without introducing congestion;
- providing stable saturation throughput.

Chapter contents

6.1 Analysis and motivation

This section analyzes some limitations of previous work that motivated the development of LIAN: the counter-based traffic detection and the impact of short non-minimal paths. First, regarding counter-based adaptive routing, it identifies how forwarded-traffic counters measure actual *carried traffic* instead of *offered traffic*,[1] which significantly reduces throughput after the saturation point. Next, regarding the length of non-minimal paths, it studies the impact of short non-minimal paths on different traffic patterns.

6.1.1 Traffic counters measure carried traffic

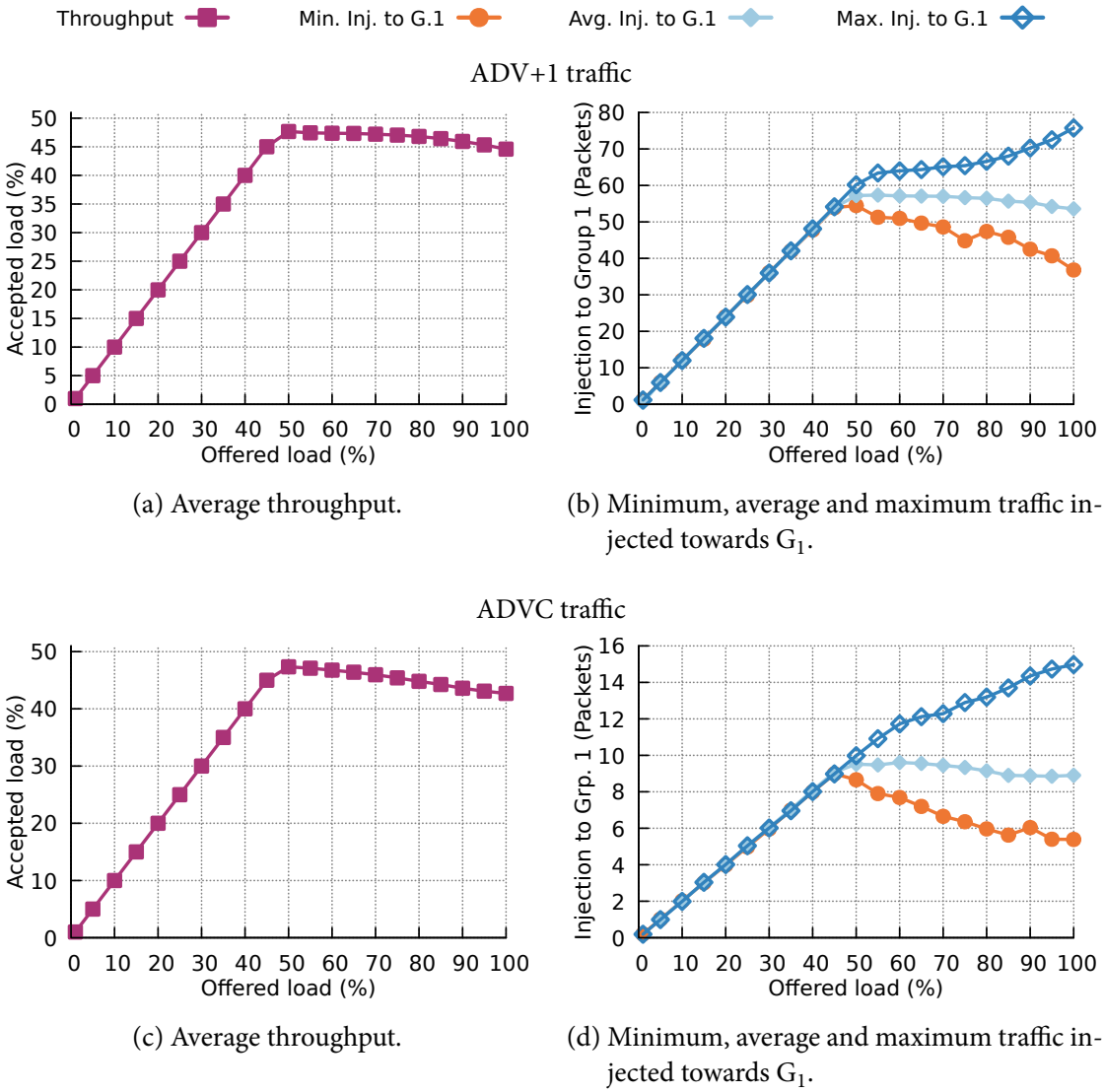
Routers for HPC systems maintain an extensive number of performance counters [49, 131]. The main use of them is monitoring the systems and analyzing their performance by providing information about latency, number of flits across interfaces, stalls, etc [35, 56, 8]. However, these collected statistics can be leveraged, for example, to infer the traffic pattern or to quantify the contention present in the network. Based on this information, an adaptive non-minimal routing algorithm can be designed.

Traffic Pattern-based adaptive Routing for Dragonfly networks (TPR, Faizian *et al.* [61]) relies on aforementioned traffic counters to modulate the UGAL decision between minimal and non-minimal paths at the source. TPR defines counters that measure both local and non-local intra-group and inter-group traffic and based on these counters it defines multiple regions that specify different levels of intensity or adverseness. For example, the local inter-group counters track the amount of packets for a given destination group that are forwarded from the input ports, per interval. All of these counters do not measure offered traffic,[1] instead they measure carried traffic, which is influenced by the routing algorithm, generating a cyclic dependency because they are used to influence routing. This dependency may mislead the result of the adaptive routing mechanism at saturation, as explained in the following example.

Figures 6-1(a) and 6-1(c) show the throughput vs applied load plots for a Dragonfly network under *Adversarial shift* (ADV+i) with offset $i = 1$ and *Adversarial Consecutive* (ADVC) traffic patterns using TPR. At saturation, the router queues may get completely full, transiently stalling forwarding. When this happens, traffic counters may fall bellow the "high intensity" threshold, which mislabels traffic intensity as "medium" and biases the routing function towards minimal routing. This increases the network bottleneck problem and further reduces both injected traffic and the counters for the given destination group, which preserves the mislabeling problem. Overall, this results in reduced throughput after the saturation point, as observed in Figures 6-1(a) and 6-1(c). This congestion is not uniform across all routers, as observed in Figures 6-1(b) and 6-1(d). They depict maximum, average

[1] The one that would be sent in a network with infinite resources.

and minimum traffic sent to group one per router. The difference between the highest and lowest injecting nodes grows with the offered load after saturation, leading to a significant imbalance.



(a) Average throughput.

(b) Minimum, average and maximum traffic injected towards G_1.

(c) Average throughput.

(d) Minimum, average and maximum traffic injected towards G_1.

Figure 6-1: Average throughput and minimum, average and maximum value for local injection to group 1 for all switches within group 0 under ADV+1 and ADVC traffic patterns using TPR routing in a Dragonfly network with *h*=6.

6.1.2 Impact of Valiant phase A path length

As explained in Section 2.3.2, multiple VLB phase A policies can be used to generate the path for phase A which precedes the minimal path *lgl* of phase B. They were previously presented in Table 5-1. This section explores the impact of VLB phase A path length in a Dragonfly network. Note that due to VLB being implemented as *restricted* (see Section 5.2 in p. 105), if both source and destination routers are in the same group, paths of phase A consist on a single local hop *l*. Since this chapter is focused on global traffic, the non-minimal global

hop *g* is clearly required to remove the bottleneck in the global link otherwise caused by adversarial traffic when the destination is in a remote group. Therefore, four paths for Valiant phase A may be considered: *lgl*, *-gl*, *lg-* and *-g-*. Note that with two global hops in the path, throughput is limited to 50%.

First, it is analyzed the impact of saving the first local hop in Section 6.1.2.1. Next, Section 6.1.2.2 analyzes the impact of saving the second local hop, proposed originally for the Dragonfly Valiant load-balancing routing algorithm implementation in [108].

6.1.2.1 Impact of the first local hop in Valiant phase A path

This section analyzes the relevance of the first local hop l_1 in VLB phase A paths using an adversarial traffic pattern. Figure 6-2 shows throughput per router in a group managing ADVC traffic, using oblivious VLB and adaptive TPR routing algorithms, where VLB paths employ *-gl* and *lgl* policies for the phase A.

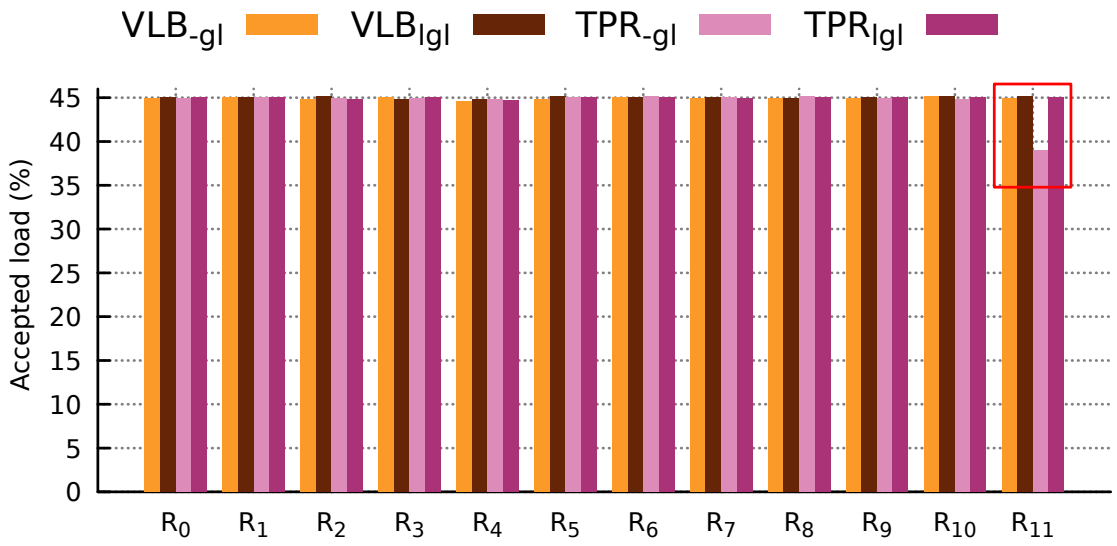


Figure 6-2: Accepted throughput for each switch of group 0 under ADVC traffic pattern, with offered load of 45%, comparing two path lengths using VLB and TPR routings.

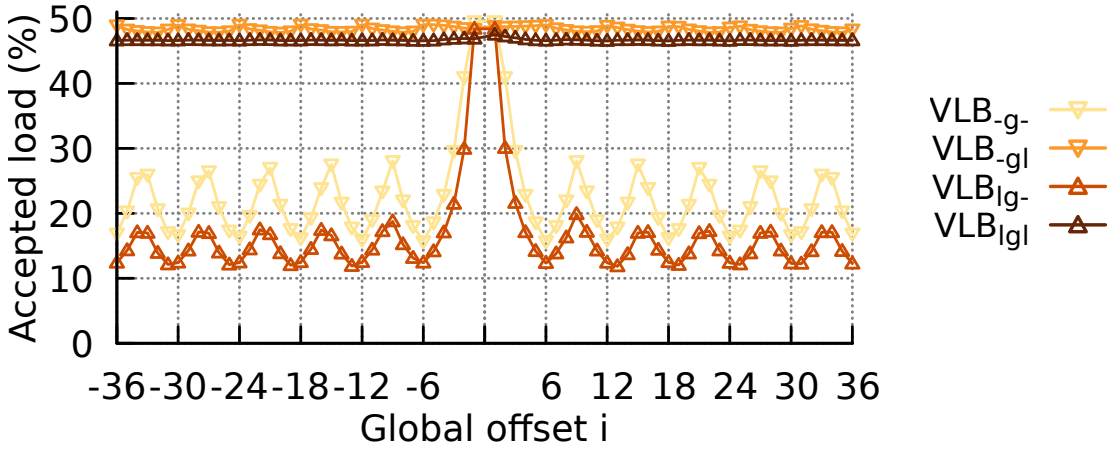
Under oblivious VLB, all injectors randomize traffic and all their traffic is accepted with both path policies. With adaptive TPR, part of the traffic from Routers 0-10 is sent minimally to switch 11 according to the ADVC traffic pattern depicted in Figure 3.2.2.1.4. This traffic, which must be forwarded by the *h* global links in Router 11 as R_{OUT} (see Figure 3-8, p. 56), interferes with the traffic sent from its own $p = h$ compute hosts. With the *-gl* path policy, R_{11} also employs the same *h* global links for its own traffic, so injection is reduced in this router causing throughput unfairness. By contrast, using *lgl* path policy the router R_{11} may employ all the global links in the group, and all the routers inject the maximum load despite the interference of minimal traffic.

6.1.2.2 Impact of the second local hop in Valiant phase A path

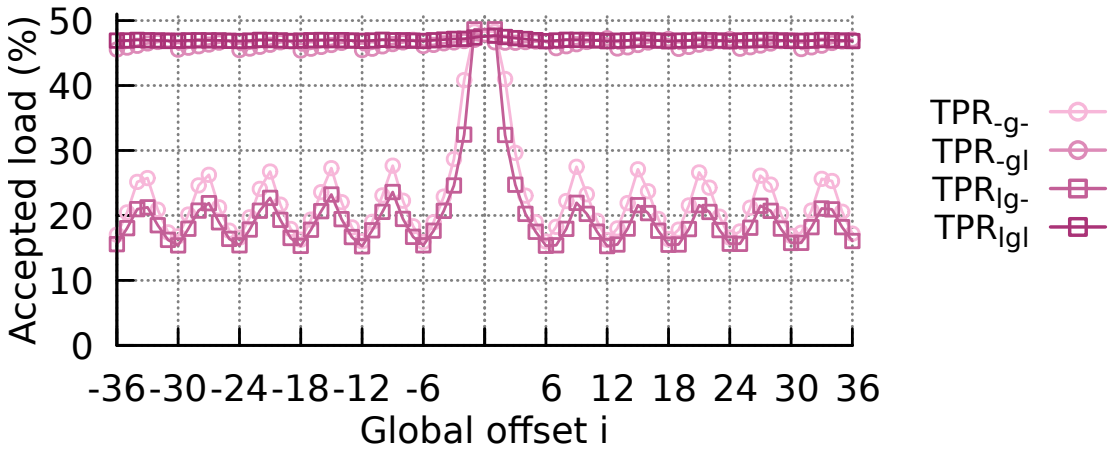
This section explores the impact of the second local hop in VLB phase A paths, based on a performance analysis of ADV+*i* traffic pattern and an analysis of the load received in local links for each variant of non-minimal path length.

This section explores the impact of omitting the second local hop l_2 in VLB phase A paths. Homogeneous traffic in which all routers inject with the same pattern is considered. This traffic can be divided into multiple ADV+*i* traffic patterns, one per global offset in the original pattern. The congestion generated by each ADV+*i* traffic independently is analyzed.

Figure 6-3 shows the accepted load under ADV+*i* traffic pattern for each possible *i* in an $h = 6$ Dragonfly network with an injection load of 0.5 phits/node/cycle. Both oblivious VLB and adaptive TPR routing algorithms are considered. There are $G = 2h^2 + 1 = 73$ groups, so $i \in \{-36, ..., -1\} \cup \{+1, ..., +36\}$, in abscissas; no value is presented for $i = 0$. Lines depict measured throughput for each of the four possible policies for VLB phase A.



(a) Oblivious VLB.



(b) TPR adaptive routing.

Figure 6-3: Average throughput with offered load of 50% on ADV+*i* traffic pattern in a Dragonfly network using oblivious VLB and adaptive TPR routings with different path lengths in phase A.

Eliminating the second local hop l_2 under these traffic patterns significantly reduces throughput: both *-gl* and *lgl* path policies almost reach the maximum throughput of 0.5 phits/node/cycle, but other ones (*-g-* and *lg-*) fall significantly lower. This motivates the use of *-g-* or *lg-* phase A path policies when the load is low to reduce latency, and *-gl* or *lgl* ones when the load is high to avoid network saturation. Also, considering ADV+*i* traffic in isolation the first non-minimal local hop does not help alleviate congestion but increases path length (and traffic load), so the throughput for *lg-* and *lgl* is respectively lower than for *-g-* and *-gl* path policies; other traffic patterns behave differently. This occurs for both oblivious VLB and non-minimal adaptive TPR routing algorithms.

The maximum throughput using *-g-* (and *lg-*) phase A paths is not constant: in these examples, it varies with the offset of the ADV+*i* traffic pattern, with minimum values for *i* being multiples of *h*, and maximum result for intermediate values of *i*. Next, an approximate model is introduced in order to estimate this maximum throughput. It is considered ADV+*i* traffic with $i > 0$; negative offset is analogous because the *palmtree* global link arrangement, described in Section 2.2.1, is symmetric. The analysis focuses on the location of the intermediate router, R_{ROOT}. Figure 3-6 shows a group which behaves as the intermediate group for non-minimal packets received from other groups, using *-g-* or *lg-* paths. The load on local links in this intermediate group is analyzed next.

Consider $ADV+i = ADV+(k \cdot h+m)$ traffic with $0 \leq m < h$ and $k \geq 0$; negative values are symmetric. With the palmtree global link arrangement employed in LIAN, traffic received in the intermediate group through router R_i[2] needs to leave to the destination group through router R_{i-k} or R_{i-k-1} $(mod\ 2h)$. Therefore, the traffic received in router R_i is forwarded by only two local links l_{-k} and l_{-k-1} in the l_3 hop. Specifically, flows received by m global links leave through local link l_{-k-1} and the remaining traffic from $h-m$ links is forwarded by local link l_{-k}.[3] The largest of m and $h-m$ determines which type of local link saturates first and gives an upper bound for accepted load, since both l_{-k} and l_{-k-1} can only accept 1 phit/node/cycle. $ADV+(k \cdot h)$ traffic ($m = 0$) provides the lowest throughput since local link l_{-k} receives the aggregated throughput of all the h incoming global links in R_i, limiting throughput to $\frac{1}{h}$ on average. The highest throughput is obtained when the global offset is the intermediate value between $k \cdot h$ and $(k+1) \cdot h$ because traffic is evenly distributed between both local links. Note also that $k = 0$ implies that part of the traffic leaves through l_{-1} and part of the traffic leaves directly without any local hop (there is no l_{-0}), reducing congestion.

[2]Section 2.3.2 denotes the router at the intermediate group that receives the packet from the source group as R_C on the VLB phase A. However, here it is interesting to identify it through its position within the group.

[3]Considering the groups numbered in counter-clockwise order in a circle, like in Figure 2-7(b); This result is not totally accurate by one unit when the intermediate group G_{ROOT} is between the source G_S and destination group G_D ($G_S \leq G_{ROOT} \leq G_D$). The source of that one unit deviation can be seen in Figure 6-6 because there are two rounded group indexes, which correspond to the negative and positive intermediate-local counters (IL_{-h}, IL_{+h}), adjacent in the middle due to the fact that $-h^2$ and $+h^2$ are different groups. This has been deliberately ignored in this analysis because it would make the model more complex with a negligible impact on performance. However, it explains the tiny drift of the minimal throughput values observed in Figure 6-3 for large offsets.

For example, Figure 6-4 depicts the case of $ADV + 8 = ADV + (1 \cdot h + 2)$ traffic using -*g*- as VLB phase A path in an $h = 6$ Dragonfly. Traffic from $m = 2$ global links is forwarded through local link l_{-2} whereas traffic from $h-m = 6-2 = 4$ global links is forwarded through local link l_{-1}. This limits throughput to 1/4 = 25% under this traffic. The results presented in Figure 6-3 are close to this limit. Note that the previous analysis does not account for the local hops in the source and destination groups, which reduce measured throughput below the previously calculated bound. Accounting for this traffic is practically not feasible.

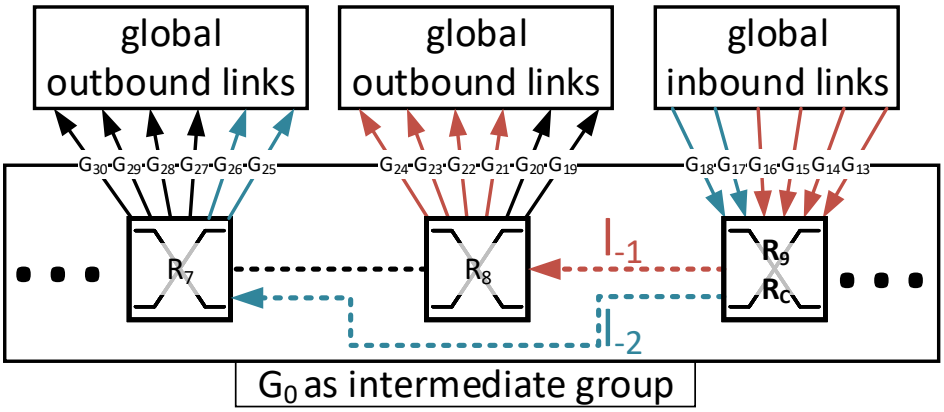


Figure 6-4: Bottleneck at local links of the intermediate group under ADV+8 traffic pattern in a Dragonfly topology with *h=6*.

6.2 LIAN: Latency-Improved Adaptive Non-minimal routing for Dragonfly networks

This section introduces *LIAN: A Latency-Improved Adaptive Non-minimal routing algorithm for Dragonfly networks*. After an initial overview, extended counters are detailed in Section 6.2.2 and the non-minimal path selection is presented in Section 6.2.3.

6.2.1 LIAN overview

LIAN can be classified as a *non-minimal source-adaptive local congestion-aware* routing algorithm. Source router computation selects a non-minimal path for each packet and determines if such packet should be sent following a minimal or non-minimal path. The non-minimal path is selected based on a *restricted* version of VLB routing algorithm[4] that can skip each of the two local hops l_1 and l_2 in phase A path. Then, the selection between minimal and non-minimal path relies on the occupancy of each path, based on a variant of UGAL.

[4]Extending the RVLB, introduced in Section 5.2, to be able to select R_{ROOT} intermediate switches that fit in routing paths skipping l_1, l_2 or both local links.

In both cases, decisions depend on an estimation of the current *offered* traffic, which relies on traffic counters.

Traffic counters combine information from injected traffic per interval and the instantaneous amount of traffic in the injection queues, as explained in Section 6.2.2. Since they extend the simple implementation of a traffic counter, these are denoted as *extended* counters. LIAN employs two sets of extended counters per router denoted as: *global* and *intermediate-local* counters. Global counters are employed to determine whether it is safe to skip the first local hop l_1 and to modulate UGAL, whereas intermediate-local counters are used determine the same for second local hop l_2.

First, $2h^2$ *global* counters $\{G_{\pm1}, G_{\pm2}, ..., G_{\pm h^2}\}$ per router measure the offered traffic towards each of the $2h^2$ remote groups. Their values are obtained from the router injection ports. Source routers determine when to skip l_1 based on these counters, as defined in Section 6.2.3.1.

Second, $2h$ calculated[5] *intermediate-local* counters $\{IL_{\pm1}, IL_{\pm2}, ..., IL_{\pm h}\}$ estimate the load that would be received on the local links in intermediate groups if the non-minimal routing omitted the l_2 local hop (*-g-* or *lg-* phase A path policies) as introduced in Section 6.1.2.2. Each router maintains one intermediate-local counter IL_i per type of local link l_i departing from it, with the terminology introduced in Section 2.2.1. Each of these counters is derived from the values of a subset of the global counters, as defined in Section 6.2.3.2.

Based on the values of the global counters and intermediate-local counters, LIAN determines a suitable policy for VLB phase A, selecting a random intermediate router that fits with such non-minimal path. Finally, the value of the global counter associated to the destination of the packet, which represents traffic intensity, is used to modulate UGAL. It biases the result towards minimal or non-minimal routing by modifying the threshold parameter T in equation 2-1. Three load levels are defined, each with its own threshold T.

6.2.2 Traffic estimation using extended counters

Section 6.1.1 identifies the limitations of estimating traffic using the amount of packets that are forwarded from each injection queue, or injected into the network. This section introduces an alternative mechanism which considers both newly *injected* packets (inj_i) and packets transiently *stored* in injection queues ($stor_i$).

If *stored* packets were considered, the corresponding global counter would be increased each time a generated packet entered an injection queue, and it would be correspondingly decreased when the packet leaved the queue. *Stored*-based counters have a very small average value before the saturation point, and grow quickly when traffic load exceeds this load, because traffic cannot be delivered as fast as it is generated and packets get stalled in the input

[5]Computed directly from the values of some *global* counters. They can be seen as SQL non-persistent calculated columns.

queues. Thus, they provide useful information under saturation. By contrast, *stored*-based counters require buffer filling, so they do not accurately reflect the traffic pattern before saturation, and they react slowly to traffic changes.

LIAN employs *extended* counters, which combine information from both *injected* and *stored* packets. The dynamic range of the original *injection*-based counters depends on the sampling interval. By contrast, the range of *stored*-based counters depends on the size of the input buffers. For this reason, the calculation of combined counters relies on a parameter w used to weight the different types of counters, as follows:

$$extended_i = inj_i + w \times stor_i. \tag{6-1}$$

The parameter w needs to be set so that: 1) the most benign traffic under saturation is not confused with adversarial traffic patterns; and 2) adversarial traffic patterns after saturation is not mistaken with medium or low-intensity traffic, as seen in Section 6.1.1 and analyzed previously in Section 4.6.3.

6.2.3 Non-minimal paths in LIAN

This section presents the mechanisms used by LIAN routing algorithm to determine when and how to shorten non-minimal paths, based on estimations of traffic.

6.2.3.1 Global counters and first local hop

Section 6.1.2.1 discussed the necessity of the first local hop l_1 in non-minimal paths. In particular, consider a shortened non-minimal path without l_1 ($-gl_2$ or $-g-$). Since the first hop is global, the selection of the intermediate group is limited to those groups directly connected to the source router, so the packet is directly forwarded using one of its global links with non-minimal routing. When minimal paths also employ these global links, there may be not enough resources to accommodate traffic from other routers, such as the minimal traffic from other links observed in Section 6.1.2.1.

LIAN avoids this problem with a simple approach: it uses the first non-minimal local hop l_1 when there are several global links in the source router that receive too much *minimal load*; *minimal load* denotes the load that these links would receive if minimal routing is used. This is obtained from the extended global counters discussed in Section 6.2.2. Specifically, each router tracks the global counters associated to the remote groups directly connected to it. If *several* of them (a parameter denoted as *Saturated Global Counters threshold*, *SGCth*) receive a value exceeding a given *Global Counter Saturation threshold* (*GCSth*), then l_1 is used, this is, the non-minimal intermediate group is selected without restriction. Otherwise, the non-minimal path intermediate group is restricted to those groups directly connected, i.e., l_1 is not used. An example of LIAN computation regarding the first local hop l_1 for a specific situation is shown in Figure 6-5.

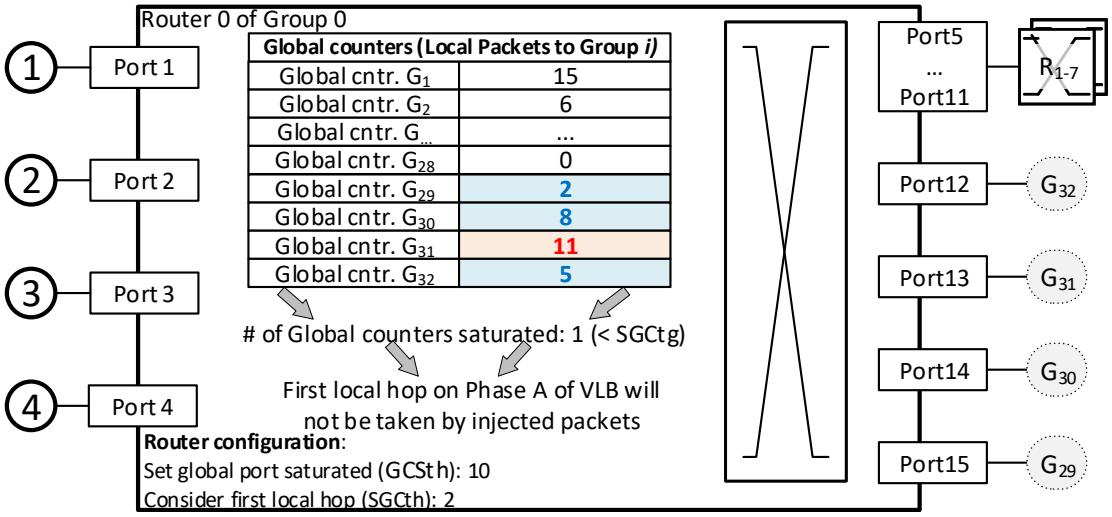


Figure 6-5: Example of LIAN decision about l_1 hop in a router R_0 of a Dragonfly network with *h=4*. The picture shows input and output ports, LIAN configuration thresholds, global counters and the *LIAN* decision.

6.2.3.2 Intermediate-local counters and second local hop

This section details *intermediate-local* counters, which estimate the extent of intermediate-group local-link congestion with the current traffic, and then how to use them to determine when it is safe to skip l_2 in the routing. Section 6.1.2.2 explains how skipping the second local hop (l_2 in phase A) in non-minimal paths concentrates traffic in certain local links used as the l_3 hop, causing pathological congestion. Specifically, for traffic with global offset $k \times h + m$, with $0 \leq m < h$, hop l_3 exclusively employs local links of type l_{-k} or l_{-k-1}, and it uses them proportionally to $(h - m)$ and m respectively.

LIAN employs $2h$ intermediate-local counters per router, one per each minimum value in the -*g*- phase A path throughput plot in Figure 6-3(a), i.e., one per each possible global destination offset $ADV + (k \cdot h + m)$ with $m = 0$. Each of these counters estimates the load imposed in one type of local link l_i, should the l_2 hop be omitted from a non-minimal path.[6]

Since a packet with global offset $k \times h + m$ may contribute to congestion in links l_{-k} or l_{-k-1}, LIAN accounts for it in both corresponding counters IL_{-k} and IL_{-k-1}, proportionally to their *probability* of use $\frac{h-m}{h}$ and $\frac{m}{h}$ respectively. Therefore, flows with different but close global offset contribute to congestion in the same type of local links: All $2h - 1$ flows with global offset $(k-1) \times h + 1$ to $(k+1) \times h - 1$ contribute to congestion in links of type l_{-k}, and have to be accounted for in the intermediate-local counter IL_{-k}.[7]

[6]Note that both minimums associated to global offsets $\pm h^2$ correspond to saturation of the local link l_{-h}; these offsets are consecutive due to the modulo function, and both appear because of the aforementioned case of choosing the intermediate group G_{ROOT} between the source and destination groups, which is more frequent for large offset values.

[7]These flows also affect counters IL_{-k+1} and IL_{-k-1}.

Intermediate-local counters do not need to be calculated from each packet from the injection queues, since global counters already track the offered load to each destination group (i.e., global offset). Therefore, intermediate-local counters are directly calculated from their values G_i. According to the previous discussion, each intermediate-local counter IL_i is simply obtained with the following sum:

$$IL_{-k} = \sum_{j=-(h-1)}^{+(h-1)} \left(\frac{h-|j|}{h} \right) \cdot G_{k \cdot h+j \ (mod\ 2h^2+1)}. \tag{6-2}$$

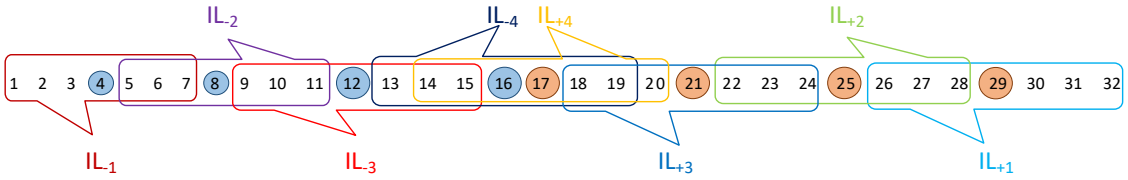


Figure 6-6: Intermediate-local counters $\{IL_{\pm1}, IL_{\pm2}, ..., IL_{\pm h}\}$ in a router of G_0 for a Dragonfly network (h=4) using the palmtree global link arrangement .

Figure 6-6 depicts intermediate-local counters of any router of G_0 in an $h = 4$ Dragonfly network. For example, in the same $h = 4$ Dragonfly network the derived counter IL_{-3} is obtained from global counters as follows:

$$IL_{-3} = \frac{1}{4}G_9 + \frac{2}{4}G_{10} + \frac{3}{4}G_{11} + \frac{4}{4}G_{12} + \frac{3}{4}G_{13} + \frac{2}{4}G_{14} + \frac{1}{4}G_{15}.$$

Once they are calculated, intermediate-local counters are used to evaluate at injection if the non-minimal path omitting l_2 is feasible, or if it introduces congestion and l_2 should be included, as follows. For each packet sent to a destination group with global offset $k \times h + m$, the source router checks its intermediate-local counters IL_{-k} associated with the destination global offset and if none of them exceeds an empirical threshold (*Intermediate-Local Congestion threshold*, *ILCth*), then congestion is not relevant and l_2 is skipped from phase A path; otherwise, l_2 is included in phase A to avoid congestion.

6.3 Evaluation

Firstly, this section presents the particular simulator configuration for the evaluations performed in this chapter and then, the performance results of the proposals. Next, *extended global* and *intermediate-local* counters are evaluated in Section 6.3.2. And finally, the introduced LIAN routing algorithm is compared to oblivious and adaptive routing algorithms in Section 6.3.3 using steady-state and transient traffic patterns.

6.3.1 Simulator configuration

The simulation experiments designed to evaluate the performance of the proposals introduced in this chapter have been carried out according to the methodology explained in Chapter 3. Steady-state and transient traffic patterns presented in Section 3.2.2 are employed for the evaluation experiments. The network simulator mimics the behavior of LIAN routing algorithm as described in previous section. It employs the extended metric for traffic counters explained in Section 6.2.2. The *intermediate-local* counters introduced in Section 6.2.3.2 are updated each cycle according to Equation 6-2. LIAN modulates the length of phase A of VLB paths as follows: when any of the intermediate-local counters associated to the destination of a packet exceeds threshold *ILCth*, the second local hop l_2 is considered, and any switch may be selected as intermediate random switch (R_{ROOT}); otherwise, only routers directly connected to the source group are selected. The l_1 hop in VLB phase A depends on *global* counters. For each global counter for a group directly connected to the source switch, when its value exceeds the *GCSth* threshold, the global counter is considered saturated. If the number of saturated global counters in a router exceeds the *SGCth* threshold, packets may take the first local hop in the phase A of VLB path.

Oblivious routing algorithms, *Minimal* (MIN) and *Valiant load-balancing* (VLB[8])) implemented as explained in Sections 2.3.1, 2.3.2 and 5.2, have been used because they provide the best performance under uniform or adversarial traffic patterns. Four paths with different lengths are considered for VLB phase A: *lgl, lg-, -gl, -g-*.

Piggyback (PB, [96]) is included as an adaptive reference that implements per-packet non-minimal source-adaptive regional congestion-aware routing algorithm, relying on state information for each global channel within a particular group. PB considers a global channel (*gc*) as saturated if the inequality shown in Equation 2-2 is satisfied. This information is distributed among switches of the group. PB employs *lgl* path policy for phase A and requires 4 and 2 virtual channels, in local and global network links respectively, to avoid deadlocks.

Piggyback-ACOR routing algorithm (PB-ACOR, Section 5.4.3) is also employed because it uses short non-minimal paths based on the load detected in the interconnection network. It extends the source-adaptive PB based on *ACOR*, which is introduced in Chapter 5. The *ACOR path A policy sequence* is agnostic to the traffic pattern, following the sequence $-g- \leftrightarrow -gl \leftrightarrow lgl$. PB-ACOR increases the number of hops in the phase A of VLB when packets are *blocked* several times at the head of the injection buffers. Two thresholds are used to increase (IT_1, IT_2) and decrease (DT_1, DT_2) the non-minimal path length. A *Hysteresis Cycle* (*HI*) is employed to provide stability and avoid oscillations. The path length is extended when the blocked packet counter exceeds the corresponding increase threshold, or reduced if it is lower than the decrease threshold after the hysteresis interval.

[8] Due to the performance of *RVLB-Recomp* is better or equal than VLB, as shown in the previous chapter, it is also used here as VLB. Hence, all VLB references use restricted and recomputation mechanisms presented in Sections 5.2 and 5.3.

Traffic pattern-based adaptive routing for Dragonfly networks (TPR, [61]) is based on *UGAL* and has been briefly explained in Section 6.1.1. Regarding the defined adverseness regions, each region corresponds to certain router counters values. Thresholds that separate adjacent regions are obtained empirically. When sending traffic to a certain destination, TPR modulates the UGAL parameter T based on the region associated to such destination, so that under intense adversarial traffic patterns it is more likely to forward traffic non-minimally and vice-versa. The implementation of TPR used in this chapter employs two load thresholds (LI_l and LI_h) to modulate the UGAL threshold T because in the performed simulations only the impact of local inter-group traffic is quantified. Hence, this implementation of TPR considers three traffic regions. TPR requires the same number of VCs as PB routing. The TPR's window size for each counter is 200 cycles. The counters are maintained using a circular queue logic, so, after the initial 200 cycles, counters will have precise values every cycle.

The combination of the base simulation parameters listed in Table 3-1 and the particular ones for this chapter presented in Table 6-1 determine the network parameters and the configuration for PB-ACOR, TPR and LIAN routing algorithms, unless otherwise stated during a particular experiment.

Table 6-1: Particular simulation parameters for the proposals of this chapter.

	Parameter	Value
PB-ACOR	Switch hysteresis interval	HI = 500 ns
	ACOR increase level thresholds	IT_1 = 15, IT_2 = 50
	ACOR decrease level thresholds	DT_1 = 5, DT_2 = 15
TPR	History window	HW = 200 cycles
	Counters Mode	Uses only inj_i (Eq. 6-1)
	Inter low/high injection ths.	LI_l = 3, LI_h = 5
	UGAL threshold constant T	benign=150, mix=-30, adv.=-150
LIAN	History window	HW = 200 cycles
	Inter low/high injection ths.	LI_l = 3, LI_h = 5
	UGAL threshold constant T	benign=150, mix=-30, adv.=-150
	Counters mode	Extended (Equation 6-1)
	Extended counter weigh	w = 0.1
	Intermediate-local congestion th.	$ILCth$ = 20
	Global counter saturation th.	$GCSth$ = 5
	Saturated Global Counters th.	$SGCth$ = 3

6.3.2 Extended global and intermediate-local counters

This section evaluates the impact of *extended global* and *intermediate-local* counters, introduced in Section 6.2.2 under different traffic patterns.

6.3.2.1 Extended global counters in LIAN

This section explores the use of *extended global* counters. To isolate the impact of the counter model, all mechanisms rely on the longest paths for non-minimal routing (pure VLB with *lgl* path for phase A) and global counters are used to modulate UGAL. Four different counter models are explored: *Injected* is a traditional implementation in which the counter tracks forwarded traffic ($w = 0$ in Equation 6-1); *Stored* only considers the packets that transiently stay in the input queues, and is only presented as a reference; and *Extended*, using two different weight values $w = \{0.1, 1\}$ as defined in Equation 6-1. Figure 6-7 shows the traffic counter value and throughput for each traffic pattern: ADV+1, ADVC and UN. ADV+*h*, which is omitted, is similar to ADV+1.

The *Injected* counter model reflects the limitations analyzed in Section 6.1.1: under transient network stalls and over the saturation point, input queues get full because traffic cannot be forwarded as fast as it is generated and the *Injected* counter value decreases. This situation causes an incorrect estimation of the traffic pattern, guessing that the network load is less adverse and incorrectly biasing UGAL towards MIN. Hence, the saturation throughput starts to fall as show on Figures 6-1(a) and 6-1(a) and in Figures 6-7(g) and 6-7(h).

Regarding *stored*-based counters, before saturation almost all packets are immediately forwarded and the counter has a minuscule average value. This value grows quickly when the offered load exceeds the saturation point. This happens because packets cannot be delivered as fast as they are generated, so they are received at the injection rate and are delivered at the accepted load rate, quickly accumulating at the injection queues. The aforementioned has two problems for traffic estimation: first, it is difficult to differentiate UN traffic at saturation from adversarial traffic at medium or high loads, as observed in the poor throughput curves (particularly, UN shows severe congestion after saturation); second, they only pass the traffic thresholds after the saturation point, but the routing should react before this point. Latency results in Figure 6-7 show poor results for *Stored* counter model under adversarial traffic. Therefore, *stored*-based counters alone cannot be used to estimate traffic pattern.

Extended counters combine *injection* and *stored* weighted by the parameter w, as explained in Section 6.1.1. Extended counters prevent the lack of information of using only the *injection*-based counter and avoid the problem caused on UN throughput by having only the *stored*-based counter. The weight w needs to be large enough to avoid congestion under saturation in adversarial traffic patterns (by increasing the counter values and the biasing towards non-minimal routing), but small enough to prevent the congestion in UN. Figure 6-7 presents two options for tuning the parameter w. When $w = 1$ is used, the large value of the

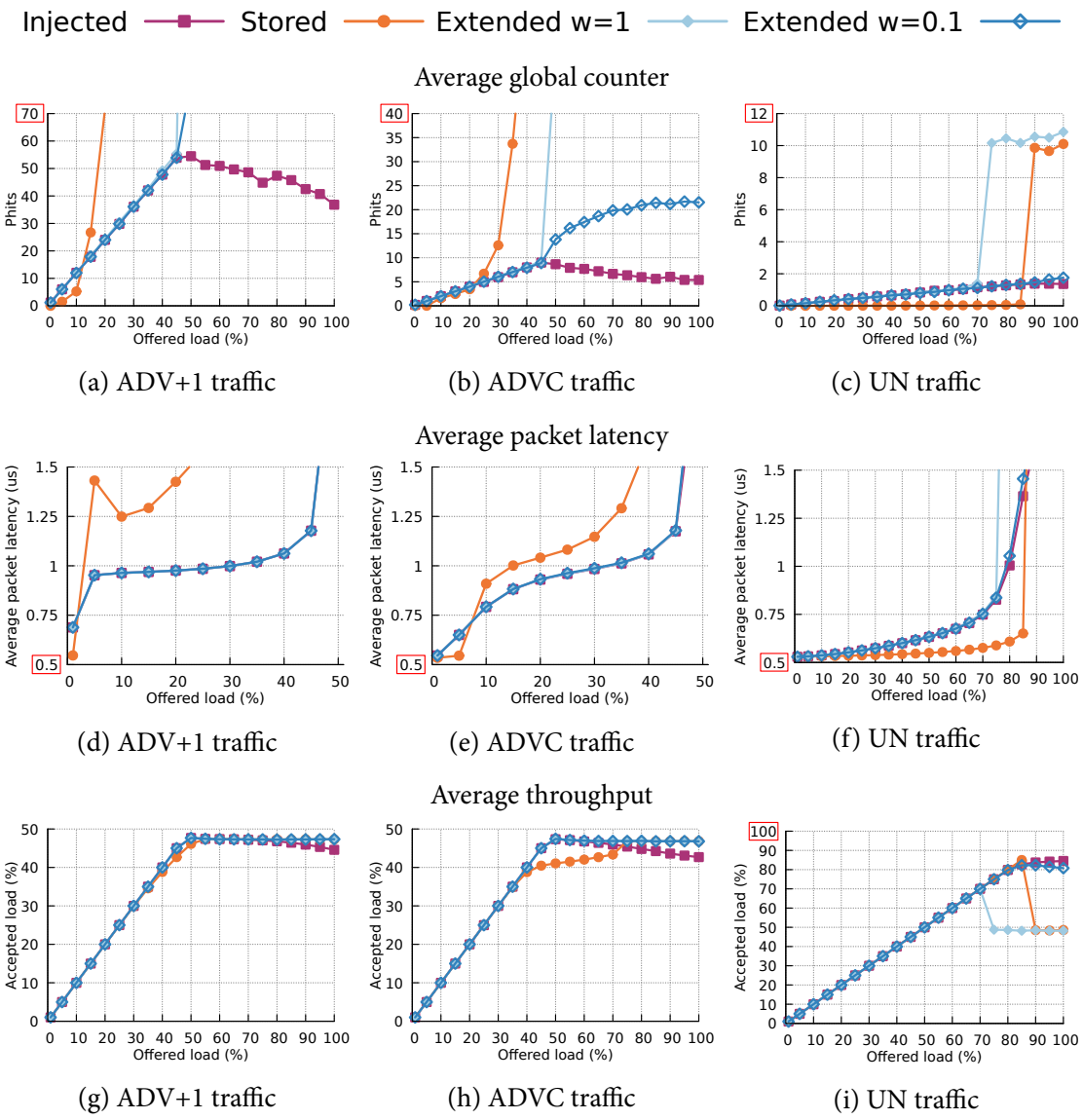
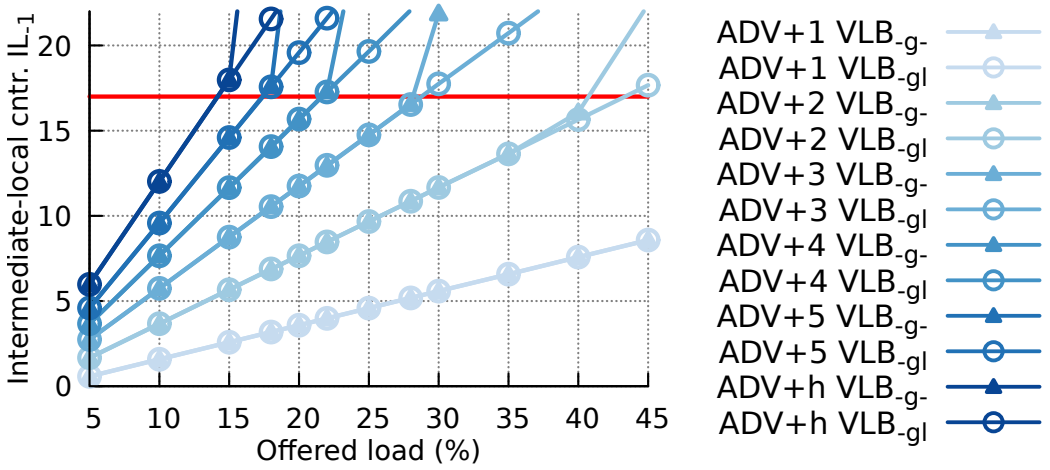


Figure 6-7: Impact of extended counters. Average global counter value, packet latency and throughput for four implementations of traffic counters and three traffic patterns.

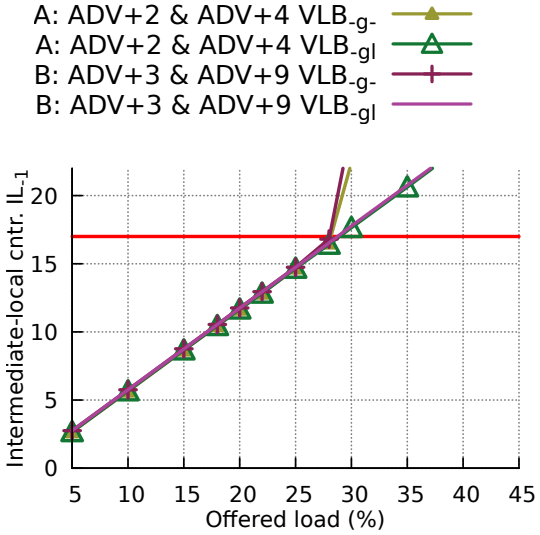
stored component produces the same undesired behavior under UN traffic pattern, because counter values at saturation become too large. A small value $w = 0.1$ lowers the extended counter to the same range of the *injection* counter alone under UN traffic pattern, as seen in Figure 6-7(c). The throughput obtained under UN by extended counters using $w = 0.1$ is similar to using *injection*-based counter alone and the throughput under ADV+1 and ADVC keeps stable after saturation thanks to the effect of the queued packets.

6.3.2.2 Intermediate-local counters in LIAN

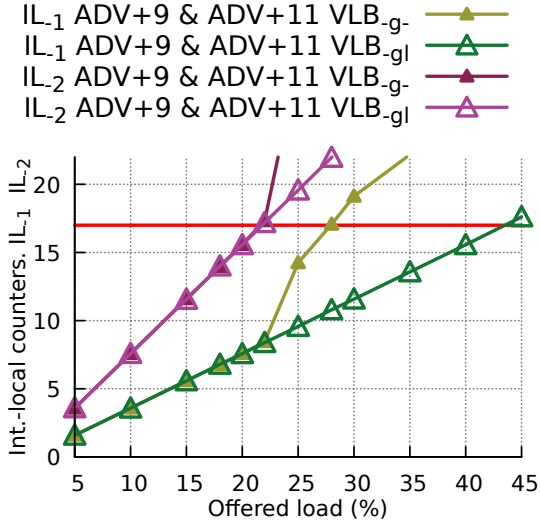
This section explores the use of *intermediate-local* counters and how they help to determine when the second non-minimal local hop l_2 is required. Figure 6-8 presents the evolution of the relevant intermediate-local counters for different traffic patterns and load. Two different non-minimal paths are considered: *-gl* and *-g-*, with and without the second local hop l_2.



(a) IL_{-1} counter value for ADV+i traffic pattern, $i \in \{1, ..., h = 6\}$ with -*g*- and -*gl* phase A paths.



(b) IL_{-1} counter value for two different pairs of combined adversarial traffic patterns.



(c) IL_{-1} and IL_{-2} counters under combined adversarial traffic patterns ADV+9 and ADV+11.

Figure 6-8: Intermediate-local counter analysis, using non-minimal paths -*g*- and -*gl*-. Saturation using -*g*- occurs approximately at the same counter value in all cases, indicated in red.

The first experiment in Figure 6-8(a) considers six adversarial traffic patterns ADV+i, with different global offset i from 1 to $h = 6$. According to the analysis in Section 6.1.2.2 and the values in Figure 6-3(a), each of these traffic patterns get a progressively lower saturation point with the phase A path -*g*-, but -*gl* is always close to the 50% limit. This occurs because -*g*- concentrates traffic on one or two intermediate local links. Intermediate-local counters track the load in such links, so their value at saturation should be the same regardless of the traffic. Results confirm the previous analysis: all the different traffic patterns saturate at different load value, but the intermediate-local counter value at saturation is similar in all cases, around 17 phits/interval. Saturation is identified because counter values using -*g*- phase A path start to grow much faster, when input queues start to fill, but when using phase

A path *-gl* they continue to grow with the same rate. Before this saturation point, counters for both phase A paths *-g-* and *-gl* match.

Intermediate-local counters application is not restricted to adversarial patterns that send traffic to a single destination group. Figure 6-8(b) presents results for a mixed traffic pattern, introduced in Section 3.2.2.1.5, that divides traffic evenly between two destination groups: groups +2 and +4 in case "A" and groups +3 and +9 in case "B". Both combinations saturate at the same range as before, around 17 phits. Additionally, the IL_{-1} counter value overlaps in both cases. Indeed, when the non-null corresponding values are replaced in Equation 6-2 using $h = 6$ both cases match, since:

$$IL_{-1A} = \frac{2}{6}G_2 + \frac{4}{6}G_4 \qquad and \qquad IL_{-1B} = \frac{3}{6}G_3 + \frac{3}{6}G_9.$$

Since traffic is divided evenly, $G_2 = G_4$ and $G_3 = G_9$, so both IL_{-1A} and IL_{-1B} grow at the same rate.Finally, Figure 6-8(c) presents an example use case, in which the traffic pattern employed divides traffic evenly between two destination groups, +9 and +11. Since both values lie between $h = 6$ and $2h = 12$, these flows will increase remote-local counters IL_{-1} and IL_{-2} as follows:

$$IL_{-1} = \frac{3}{6}G_9 + \frac{1}{6}G_{11} \qquad and \qquad IL_{-2} = \frac{3}{6}G_9 + \frac{5}{6}G_{11}.$$

Results confirm that IL_{-2} grows faster and determines the saturation point, at around 22% of the offered load. After this point, the values of both counters increase suddenly for phase A path *-g-*, since injection is restricted after local link l_{-2} saturates and packets accumulate at the input queues; this does not occur using paths *-gl*.

These results confirm that intermediate-local counters can be used to estimate the necessity of the second local hop l_2 in phase A path when using non-minimal routing.

6.3.3 LIAN performance results

This section evaluates the performance of *LIAN* compared with oblivious and other adaptive routings, considering latency, throughput, fairness and response time to traffic changes.

6.3.3.1 LIAN compared to oblivious routings

This section compares the performance of LIAN routing algorithm with oblivious routings. *MIN* has been selected as the baseline reference under *UN* traffic pattern because it achieves the best performance on both, latency and throughput. For *VLB* routing, the four non-minimal phase A path lengths considered in this chapter have been employed: *-g-*, *-gl*, *lg-* and *lgl*. VLB_{lgl} presents the best randomization to avoid any pathological congestion. However, it is at the cost of higher base latency compared with the other shorter paths. Figure 6-

9 presents average latency and throughput of *MIN*, *VLB* with the above-mentioned path lengths on phase A and the proposed *LIAN* under different traffic patterns.

Performance results under UN traffic pattern are as expected: *VLB* with different path lengths present different base latencies and a throughput close to 50%. By contrast, *LIAN* mimics *MIN* and achieves optimal latency and throughput.

Using *MIN* under adversarial traffic patterns, saturation is reached at very low load because the global links between neighbor switches become a bottleneck, and only a small part of the traffic can be delivered using the minimal routes, concretely $\frac{1}{a \cdot p} = \frac{1}{2h^2} \simeq 1.38\%$ and $\frac{h}{a \cdot p} = \frac{1}{2h} \simeq 8.33\%$ under ADV+*i* and ADVC respectively. Before these low saturation points, optimal latency is obtained using *MIN*. *VLB* raises the saturation point to different loads, depending on the non-minimal path used. *LIAN* routing raises that point near 50% in all adversarial traffic patterns presenting also optimal latency.

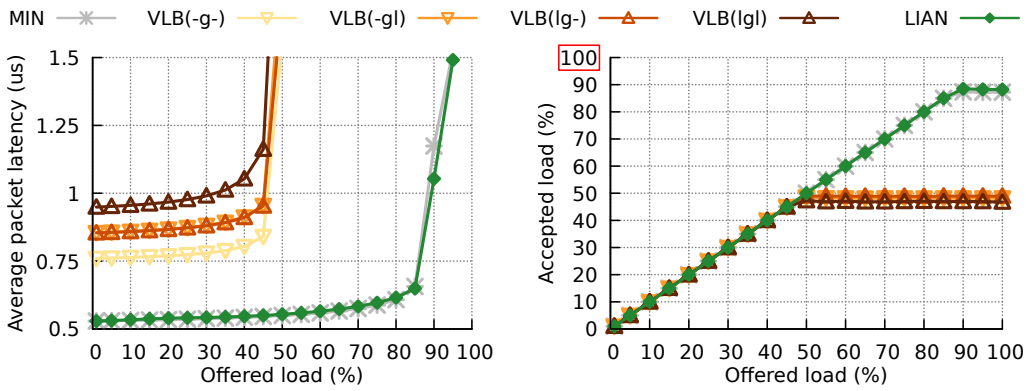
Under ADV+1, VLB variants from 1 hop (*-g-*) to 3 hops (*lgl*) in phase A obtain different base latencies, but throughput is near 50% before saturation point in all cases. After this point, only VLB_{lg-} suffers congestion. *LIAN* routing obtains optimal latency like VLB with only a global hop (*g*) on phase A, because of its latency in Figure 6-9(b), and a stable throughput very close to 50%.

Under ADV+*h*, the local link bottleneck limits the throughput up to $1/p \simeq 16\%$ using VLB phase A policies *-g-* and *lg-*. The two other VLB cases with the second local hop reach a throughput near the 50% limit. *LIAN* routing obtains a throughput close to the 50% limit and achieves an optimal latency: at different increasing loads it approximates the latency of *MIN* (up to 1.38%), VLB_{-g-} (up to 16.6%) and VLB_{-gl} (up to 50%).

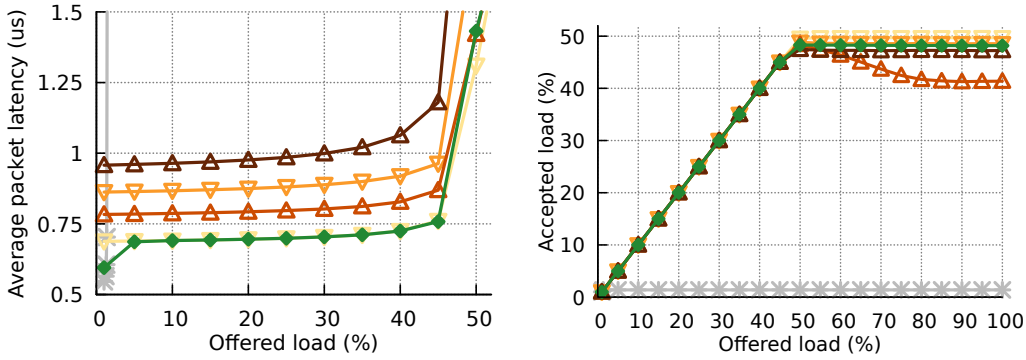
The results under ADVC follow the same trend as ADV+*h*. In this case, the saturation point of *-g-* and *lg-* is around 25%, and 50% with the second local hop. As previously, *LIAN* achieves an optimal latency because it adapts the routing to the network conditions.

Under this ADVC traffic pattern, terminals directly attached to the latest router of the group, R_{OUT}, have uneven access to the global links to be used for minimal routes. This direct connection favors a higher amount of minimally-routed traffic from those injectors, but prevents them from sending traffic non-minimally when paths *-g-* or *-gl* are used. Figure 6-10 shows the average accepted load for each switch in the group 0 of the network, under an offered load of 50% and ADVC traffic. VLB with *-g-* or *lg-* phase A paths suffers reduced throughput and pathological unfairness effects, particularly the latter. By contrast, when *-gl* or *lgl* phase A paths are used, VLB is fair because all the switches exhibit a similar accepted traffic load. *LIAN* is totally fair because all routers have the same accepted load.

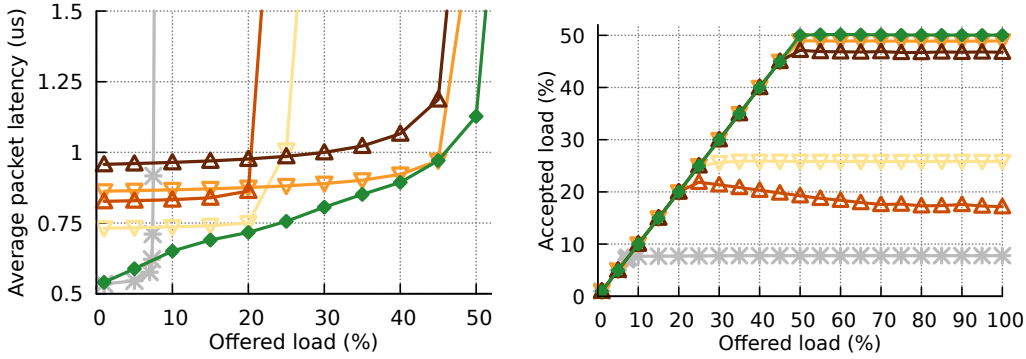
In conclusion, *LIAN* routing achieves a performance equal to or better than the best oblivious routing algorithms in both metrics, latency and throughput, for each traffic pattern analyzed. Moreover, *LIAN* does not present obvious effects of routing unfairness.



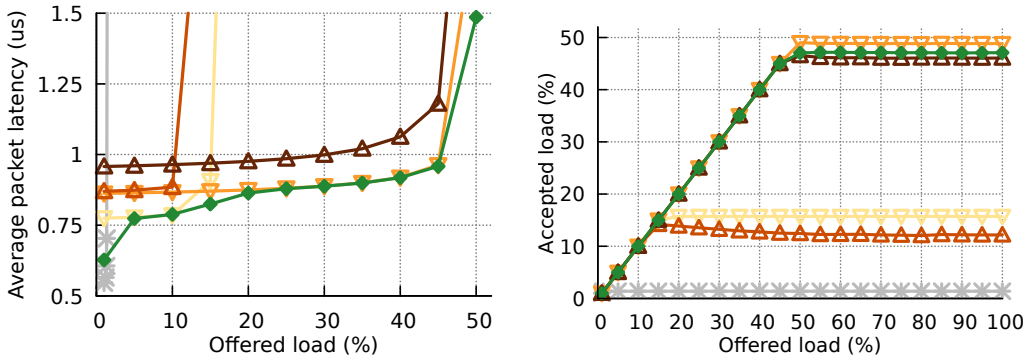
(a) Average packet latency and throughput under UN traffic pattern.



(b) Average packet latency and throughput under ADV+1 traffic pattern.



(c) Average packet latency and throughput under ADVC traffic pattern.



(d) Average packet latency and throughput under ADV+*h* traffic pattern.

Figure 6-9: Average packet latency and throughput under UN, ADV+1, ADVC and ADV+*h* traffic patterns, comparing *LIAN* with oblivious routings.

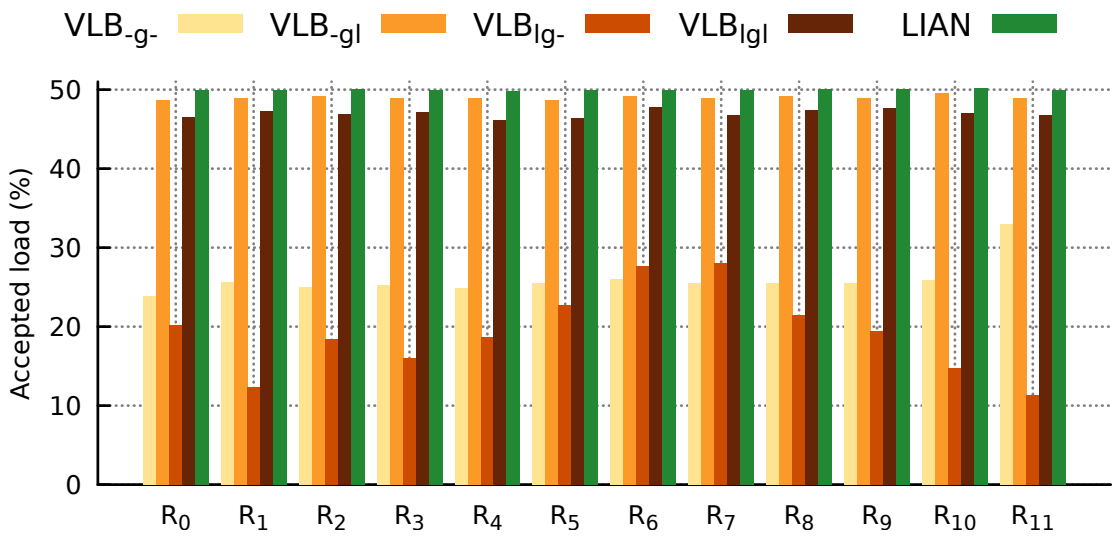


Figure 6-10: Average throughput accepted for each switch of G_0 under *ADVC* traffic pattern with an offered load of 50%, comparing *LIAN* with oblivious routings.

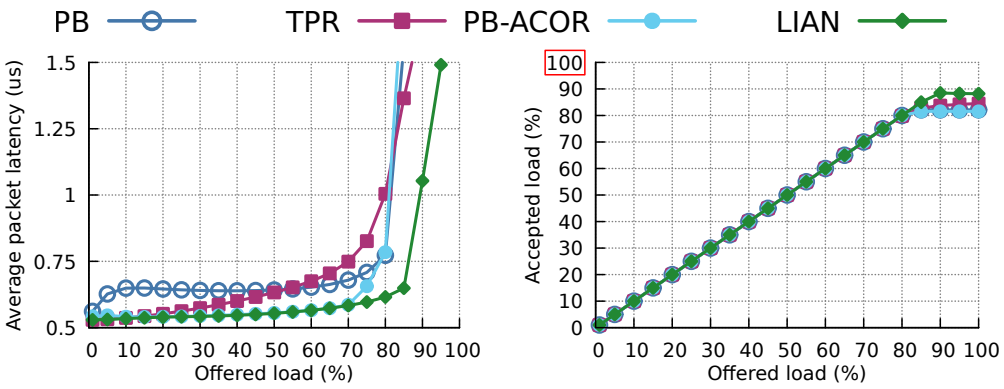
6.3.3.2 LIAN compared to other source adaptive routings

This section compares the performance of *LIAN* with other source-adaptive routing algorithms. *PB* has been selected as the baseline reference. *TPR* has been evaluated because it relies on traffic counters and modifies the UGAL parameters. *PB-ACOR* blindly adapts non-minimal path to network load for reducing latency. Figure 6-11 presents the performance of these mechanisms and *LIAN* under different traffic patterns.

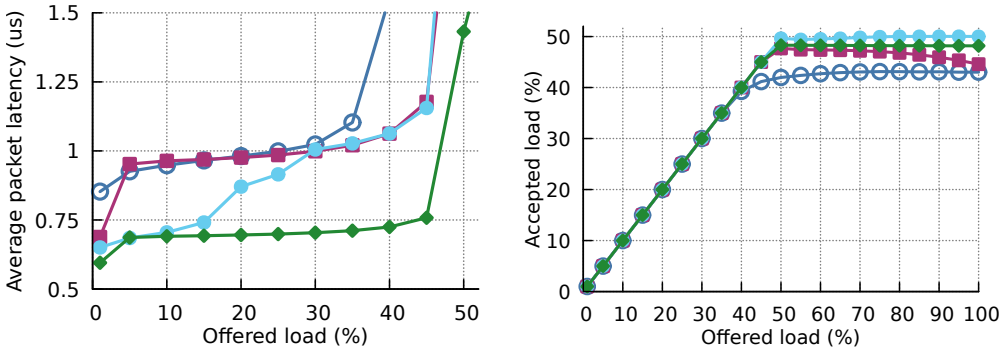
In terms of latency, both PB and TPR employ complete non-minimal paths, so under adversarial traffic patterns in Figures 6-11(b)-6-11(c) they quickly catch up with the latency of VLB_{lgl} presented earlier in Figure 6-9. Even at low load of only 10%, LIAN reduces latency by 27% over PB and by 28% over TPR under ADV+1 traffic. PB-ACOR also improves latency at low loads, but it adapts the path length to absolute network load only, whereas LIAN selects the suitable path according to the inferred traffic pattern and load. At an intermediate load of 30%, LIAN reduces PB-ACOR latency by 30.0% under ADV+1 traffic pattern. Under UN traffic pattern all routing algorithms are competitive.

In terms of throughput, TPR has a peak result close to 50% under adversarial traffic patterns in Figures 6-11(b)-6-11(c). However, after this saturation point its throughput falls, as explained in Section 6.1.1, down to the result obtained by PB, which is the lowest in our evaluations. PB-ACOR and LIAN obtain a stable throughput over saturation very close to the theoretical maximum of 50%. Again, under UN traffic all routings are competitive.

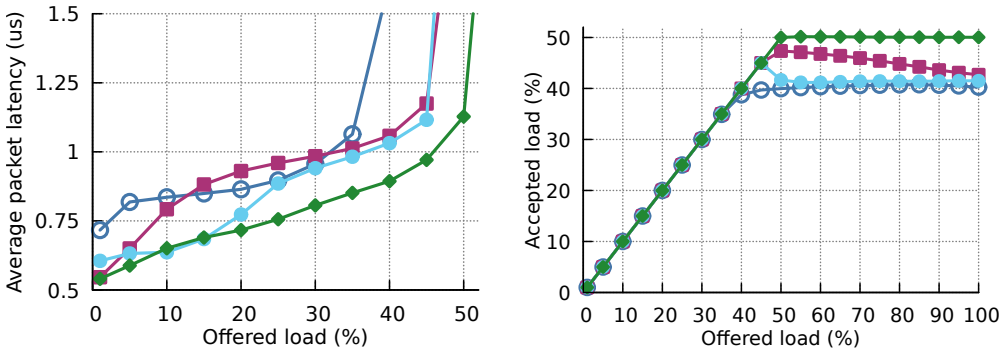
Figure 6-12 shows the average accepted load grouped by switch for the analyzed source adaptive routings under an offered load of 50%. As it is well known, PB routing suffers a pathological unfairness issue under ADVC [68]. TPR also exhibits this problem but it is less pronounced. Whereas ACOR adapting the non-minimal path length is fair (see Figure 5-6, p. 120), PB-ACOR inherits the unfairness issue of PB since it is based on the latter to select



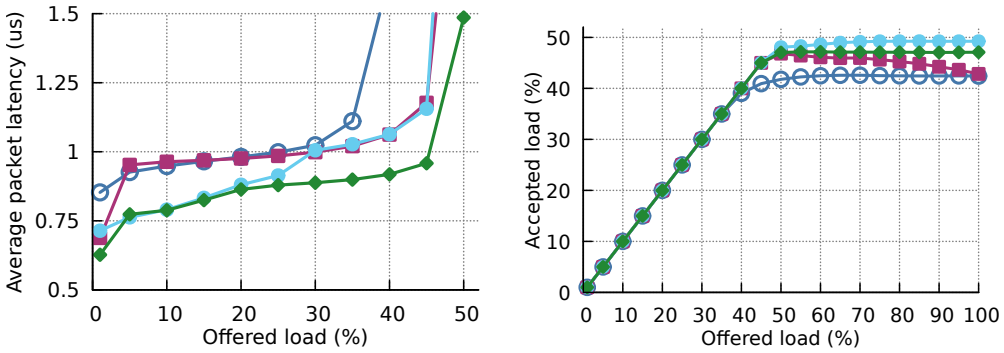
(a) Average packet latency and throughput under UN traffic pattern.



(b) Average packet latency and throughput under ADV+1 traffic pattern.



(c) Average packet latency and throughput under ADVC traffic pattern.



(d) Average packet latency and throughput under ADV+*h* traffic pattern.

Figure 6-11: Average packet latency and throughput under UN, ADV+1, ADVC and ADV+*h* traffic patterns comparing *LIAN* with other source-adaptive routing mechanisms.

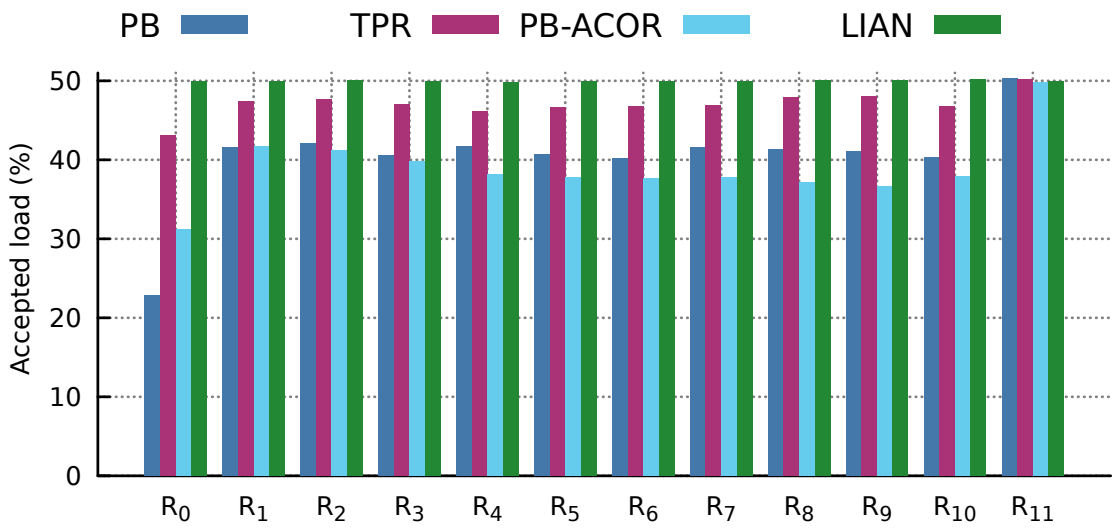


Figure 6-12: Average throughput accepted for each switch of G_0 under *ADVC* traffic with an offered load of 50% comparing LIAN with other source-adaptive routings.

between minimal or non-minimal routes. As shown, LIAN exhibits a perfect throughput fairness between all switches within the group.

In conclusion, the latency result achieved by LIAN is optimal and the throughput is very close to the theoretical maximum for each traffic pattern analyzed.

6.3.3.3 Throughput fairness and the use of the l_1 hop

This section analyzes the behavior of LIAN considering the use of the l_1 hop, according to the implementation described in Section 6.3.1. While Figures 6-10 and 6-12 present throughput fairness results under ADVC traffic pattern, Figure 6-13 breaks the accepted load for each switch by the path followed, either minimal or any of the non-minimal paths considered in this chapter.

As explained previously in Section 3.2.2.1.4, ADVC unfairness comes from the concentration of traffic in router R_{OUT} (R_{11} in the presented example) to be forwarded to the h consecutive groups. For this reason, short non-minimal paths that omit the l_1 hop should not be used by the terminals connected to the bottleneck switch R_{OUT}. This is solved by LIAN according to the explanation in Section 6.2.3.1. As shown in Figure 6-13, R_{11} identifies its global queues as saturated and it sends traffic using non-minimal paths with the l_1 hop. Other routers in the group continue using minimal paths or non-minimal paths without this l_1 hop.

6.3.3.4 Performance under transient loads

LIAN is able to adapt the routing to the live network situation by changing the VLB phase A policy used by each packet and modulating UGAL based on the inferred traffic pattern present in the network. This section evaluates these capabilities: 1) using a traffic pattern

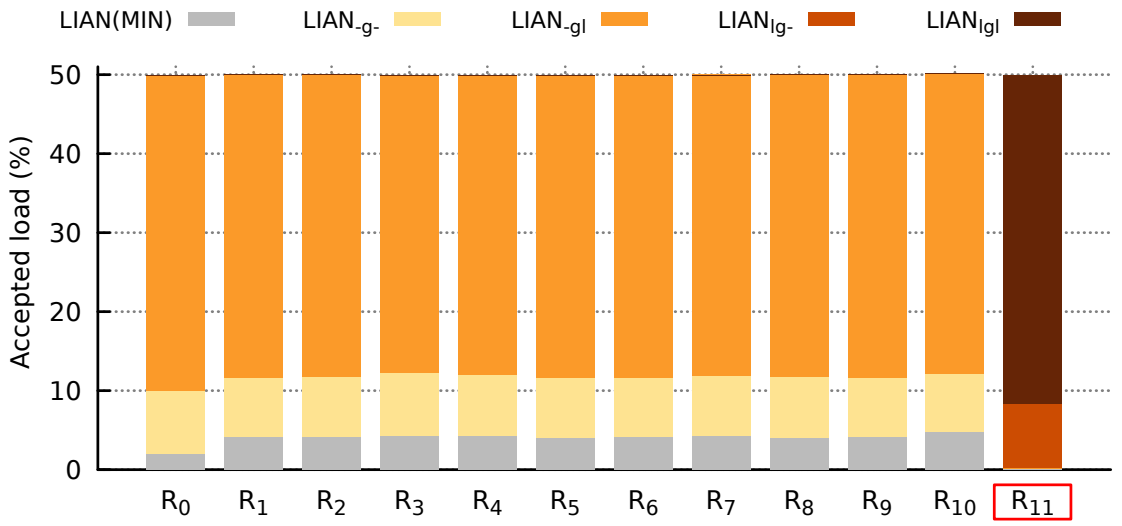


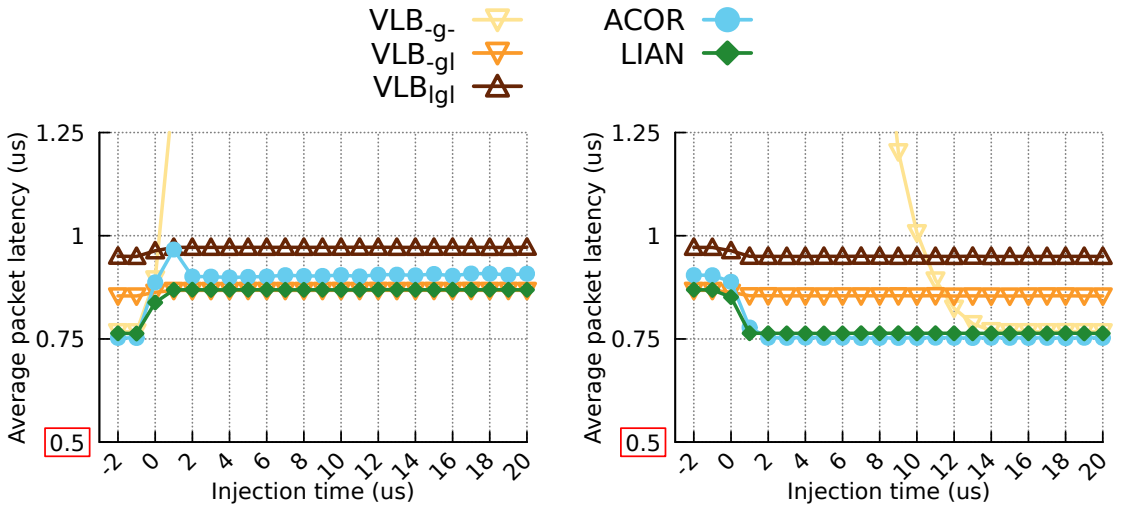
Figure 6-13: Average throughput accepted using LIAN routing for each switch of G_0 divided into MIN and the four possible VLB phase A policies under ADVC traffic pattern with a load of 50%.

with two given offered loads, which requires LIAN to adapt the phase A path policy; and 2) using two different traffic patterns that requires LIAN to decide between a minimal or non-minimal path for each packet.

Figure 6-14 shows the transition between different phase A policies due to a changing on the offered load. It displays the average packet latency under ADV+*h* traffic that changes from 5% to 25% and vice versa. These values are selected because they are supported by different phase A path policies. Three relevant *VLB* phase A policies are provided as a reference. *PB* and *TPR* are not presented because both routing algorithms apply directly VLB_{lgl} and do not adapt the path length, so there is no difference in their latency during the experiment except the change due to load variation, which can be analyzed in Figure 6-11.

When the traffic load increases, in Figure 6-14(a), the VLB_{-g-} policy is no longer competitive and the latency of VLB with this policy increases significantly. ACOR transitions from using the VLB_{-g-} policy that provides the lowest base latency at low load (5%) to a VLB_{-gl} policy which has the lowest latency at high load (25%). This change is done in less than 2 μs but it has a non-desirable peak. However, LIAN is able to adapt the phase A path followed by the packets without any transitory peak. Conversely, when the traffic load decreases, in Figure 6-14(b), VLB_{-g-} recovers a non-saturated situation after a delay higher than 8 μs. As previously, the other two VLB curves remain equally except for the minimum change due to the difference of offered load. PB-ACOR and LIAN quickly reduce the phase A path length for the packets sent non-minimally. In both cases, LIAN employs the most suitable phase A policy for each situation, and it is able to react to traffic changes from a steady situation without any transitory issue.

Figure 6-15 shows the average packet latency suffered during the transitions from UN to ADV+*h* traffic pattern and vice versa while keeping 25% offered load. Different VLB curves

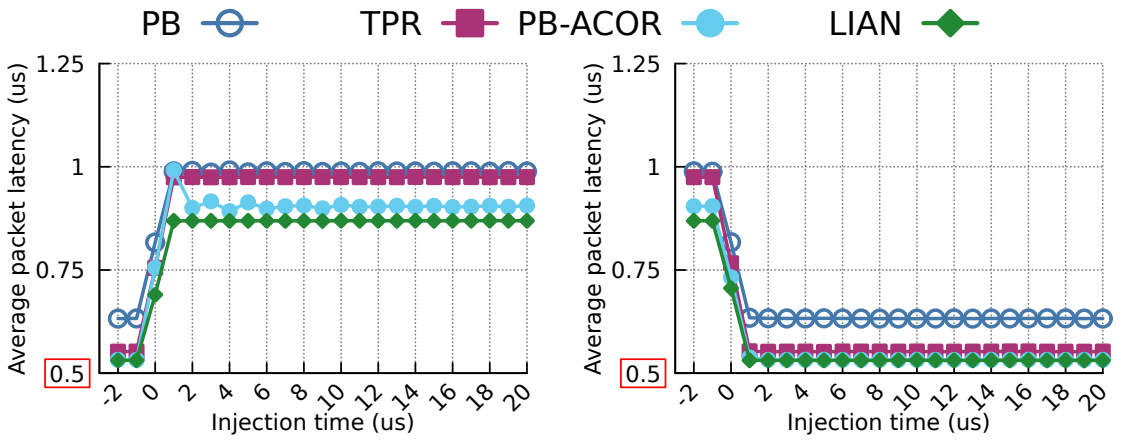


(a) From 5% to 25%.

(b) From 25% to 5%.

Figure 6-14: Average packet latency of LIAN under ADV+*h* traffic pattern with transient traffic loads. At *t=0*, load transitions (a) from 5% to 25% and (b) from 25% to 5%.

are omitted for clarity. These results can be seen for the same experiment in Figure 5-14. *PB*, *TPR* and *ACOR* are presented as adaptive references. The first two do not adapt the non-minimal path length and the latest adapt VLB phase A path as explained before. PB and ACOR shares information between the group to decide if routing a packet minimally or non-minimally based on the global links availability. However, TPR and LIAN employs only router's information to make that.



(a) From UN to ADV+*h*.

(b) From ADV+*h* to UN.

Figure 6-15: Average packet latency of LIAN under transient traffic loads with an offered load of 25%. At *t=0*, load transitions (a) from UN to ADV+*h* and (b) from ADV+*h* to UN.

When the traffic pattern changes from UN to ADV+*h*, in Figure 6-15(a), the PB's latency grows quickly because it increases the amount of non-minimally routed packets to adapt to the adversarial traffic pattern. The latency result of TPR before $t = 0$ explains that the amount of non-minimally traffic injected by TPR during the UN phase is lower than by PB.

The same can be concluded for ACOR and LIAN. After that change, these three algorithms starts to route the packets through non-minimal paths, deduced from the increment on the average packet latency. The increment of ACOR suffers a peak which was detected in Section 5.5.5.2. As it can be seen, the path length determined by TPR is equal to PB and LIAN offers the best result, keeping the latency stable and as low as possible. Conversely, in Figure 6-15(b) the behavior is similar. In this case ACOR does not suffer any transitory issue and as before, LIAN obtains the best result.

In conclusion, LIAN is able to adapt the routing to send the traffic minimally or non-minimally and to modify the length of non-minimal paths based on the network and particular router conditions.

6.4 Conclusions

Low-diameter high-radix interconnection networks, such as Dragonflies, require non-minimal adaptive routing to deal with adversarial traffic patterns due to its lack of minimal-length path diversity. Such non-minimal adaptive routing algorithms typically rely on UGAL mechanism and Valiant algorithm, which sets the length of non-minimal paths to the double of minimal ones. Traffic counters, which are widely available in actual products, contain valuable information about potential congestion points in the network.

The estimation of offered load can be used to shorten the length of non-minimal paths, provided it does not introduce additional congestion points. This is leveraged by the LIAN proposal, which combines UGAL, *extended* traffic counters to estimate offered load and a specific regular global link arrangement of the Dragonfly topology to dynamically optimize the length of non-minimal paths.

The results show that properly tuned *extended* counters correctly identify network traffic in all possible ranges, leading to accurate routing under both benign and adversarial traffic patterns. Also, *LIAN* routing algorithm achieves a performance equal to or better than the best oblivious routing for each traffic pattern analyzed. In those conditions, *LIAN* also achieves optimal latency given that non-minimal paths with the ideal number of hops are used when VLB is required to avoid congestion. This feature allows latency reductions up to 30.0% over *PB* adaptive routing algorithm. Furthermore, its throughput is very close to the theoretical maximum for each traffic pattern analyzed, which makes it very competitive compared with state-of-the-art adaptive routing algorithms. Moreover, *LIAN* does not present obvious effects of routing unfairness and its throughput is stable after saturation point.

With these features, LIAN is the first non-minimal source-adaptive congestion-aware routing algorithm that combines the desirable properties of relying solely on local information, employing the shortest path allowed by the current traffic pattern without introducing congestion and providing stable saturation throughput.

7 Related Work

This chapter makes a literature review and outlines the most notable related works to the proposals presented on this dissertation. The literature around non-minimal adaptive routing algorithms for efficient interconnection networks is extensive. In the same way as the objective of this dissertation is divided in two orthogonal sub-objectives, the related work is also divided in two sections. Hence, first, it is reviewed the related work concerning the research exposed in Chapter 4 and afterwards the related work regarding Chapters 5 and 6.

Chapter contents

7.1 Adaptive routing without credits

Motivated by the Mont-Blanc project [157] and its use of *System on Chips* (SoCs) which usually have an Ethernet network interface, the research exposed in Chapter 4 is focused on commodity Ethernet technology. Intel developed the Omni-Path [32] interconnection network architecture, which is based on Ethernet, but it has recently been discontinued and Intel will not continue the development of Omni-Path networks. However, Cray has recently proposed the Slingshot [166], its latest interconnection network technology, for the future exascale and hyper-scale systems. This architecture is based on high-radix Ethernet switches and some specialized custom Ethernet protocols to maintain the compatibility with standard Ethernet network devices and to improve its performance resulting in a suitable technology for *High-Performance Computing* (HPC) workloads. To do that, it upgrades or removes some limitations of Ethernet, such as the minimum packet size or the inter-packet gap, and adds some features to provide resiliency at different levels. An alternative vendor-independent interconnection technology frequent in HPC is InfiniBand [153]. *Mellanox* hardware with switches arranged in a Fat-tree topology is the most commonly used InfiniBand implementation, such as in the Summit [184] and Sierra [119] supercomputers [175]. InfiniBand switches provide low-latency lossless interconnection, with paths defined by a central *Subnet Manager* analogously to OpenFlow controllers. InfiniBand *subnetworks* are limited to 48K compute hosts because logical addresses are 16 bits with one fourth of the address space reserved for multicast. However, they have no other inherent scalability limitations. A study of adaptive routing strategies in InfiniBand networks can be found in [55]. However, InfiniBand *host channel adapters* are not typically found in embedded SoCs.

Section 4.3 has already discussed alternative mechanisms to achieve scalability with commodity Ethernet switches, such as *Virtual extensible Local Area Network* (VXLAN, [124]) encapsulation or layer-2 multipath overlays such as *Shortest Path Bridging* (SPB, [90]), *Transparent Interconnection of Lots of Links* (TRILL, [151]) and vendor variants such as Brocade's *Virtual Cluster Switching* (VCS, [37]) or Cisco's FabricPath [45]. These mechanisms still require to maintain in access switches large CAM tables with as many entries as compute nodes in the network, and the required encapsulation can hinder path latency. Packet header rewriting has been employed to provide location-dependent identification in several works, such as Portland [143], MOOSE [163] or In-Packet Bloom Filter [123].

Due to the lack of flow-control credits in Ethernet, proposals in Chapter 4 are based on other signals. Firstly, *MAR-bP* relies on link-level flow control Ethernet's pause messages present in 802.1Qbb [88]. However, this schema in a Dragonfly topology requires the propagation of the congestion to the router outputs to be able to sense it. To overcome this lack of global knowledge and notify the information about congestion, without spreading it, to the router that can resolve, a explicit signal [195] can be considered. Particularly, *QCN-Switch* introduced in Section 4.6 relies on *Explicit Congestion Notification* (ECN) messages

which have been used in commodity networks for a long time. IP has ECN bits for explicit congestion notification [158]. These bits are set when switch queues exceed a given threshold, possibly following a *marking* policy such as RED [65]. TCP may react to packets with the ECN source flag set by decreasing its congestion window, which throttles injection. A similar approach with a single-bit forward notification is used in InfiniBand [152], Intel Omni-Path [32] and *RDMA over Converged Ethernet* (RoCE, [92, 130, 131]), the latter being encapsulated over UDP/IP. When packets are received with the congestion flag active, the destination generates a response notifying the presence of congestion in the forward path. *Quantized Congestion Notification* (QCN, [89]) generates backwards congestion notification messages at layer 2, which include a multi-bit feedback value computed from both the current queue occupancy and from the increase or decrease rate. *Data Center TCP* (DCTCP, [11]) also estimates the *amount* of congestion based on the count of ECN flags measured during a given interval, which typically is equivalent to the TCP *round-trip time*. QCN-Switch concretely leverages the ECN messages from QCN already present in the network to adapt the routing. However, another alternative can be the use of specific messages [195] as it is done by Shpiner *et al.* [170].

Adaptive routing and *Software-Defined Networking* (SDN) concepts have been introduced in Sections 2.3 and 2.6 and considered through the research exposed in Chapter 4. Proposals that employ adaptive routing using SDN include Hedera [62], ElasticTree [81] or MicroTE [27]. All these proposals rely on per-flow load estimation, with an excessive latency for HPC applications as discussed in Section 4.2. A more detailed discussion on large flow recognition is presented in [115], considering several alternatives based on packet sampling or inline datapath measurement. By contrast, HPC implementations which support packet-by-packet adaptive routing have been implemented in or proposed for multiple topologies, such as folded-Clos [109, 164], Flattened Butterflies [110, 9] or Dragonflies [108, 96, 74]. Their selection between minimal or non-minimal paths typically relies on a comparison between the link-level flow-control credits of both outputs, which are not available in commodity Ethernet switches. However, Sensi *et al.* [166] have proposed the *request queue credits* in its Slingshot interconnection network architecture to be able to apply a source-adaptive routing algorithm, which selects the least congested path between four minimal and other four non-minimal paths, over their Ethernet-compatible Rosetta switches. Additionally, such packet-by-packet adaptive routing is typically not supported in commodity Ethernet switches since it permits packet reordering, which significantly interferes with *TCP fast retransmit* [190].

In the context of DC or HPC based on commodity technology, which is the starting point of Chapter 4, already mentioned proposals have been introduced for load balancing like DevoFlow [137], which derives from OpenFlow, Planck [159] architecture that employs port mirroring to extract network information and apply this knowledge to traffic engineering and the work proposed by Kandula *et al.* [99] which avoids the disturbing packet reordering

while it is able to balance the load dynamically by its introduced *Flowlet Aware Routing Engine* (FLARE) traffic splitting algorithm. At more basic level, adaptive routing is supported in that context in multiple technologies, at layer-3 such as IP or layer-2 such as TRILL [196], typically using *Equal-Cost MultiPath* (ECMP, [83]) which preserves in-order delivery by consistently mapping packets from the same flow to the same path. In such mechanisms, forwarding tables include multiple matching entries for a given destination, and the selection process is typically performed using a hash of some fields, such as the addresses, of the packet. While this guarantees that all packets from the same flow follow the same path, it does not consider non-minimal adaptive routing nor is modulated by the congestion level. Even more basic, in Cray Cascade (XC) systems [60] minimal routing algorithm is used for traffic that need to avoid packet reordering, and in IBM PERCS system [15] the programmer is who decides whether minimal or non-minimal routing is employed.

Minkenberg *et al.* [136] introduced the use of ECN messages to adapt traffic in data centers. Their proposal differs from the approach introduced in Chapter 4 in two fundamental aspects. First, they do not consider non-minimal routing, with the increased load introduced by non-minimal paths and the associated positive feedback loop. Second, they do not consider a probability for each available path, nor a recovery mechanism to restore minimal routing when congestion disappears. Instead, they consider fixed time intervals, and routing information is reset on each interval by discarding its current status and reverting to minimal traffic.

The mechanisms proposed in Chapter 4 adapt the traffic based on *pause* link-level flow-control frames (Section 4.5) or based on *snooped ECN messages* which modify the probabilities of minimal paths in the routing table (Section 4.6). *Universal Globally-Adaptive Load-balanced* routing algorithm (UGAL, [172]) selects at injection and on a packet-per-packet way between minimal or *Valiant load-balancing* routing algorithms, but it relies on flow-control credit counters and requires regional information. More elaborate mechanisms improve this non-minimal routing decision, both at the source or in-transit [96, 74]. Piggyback [96] and the *averaging* proposal in [192] consider the average occupancy (or credit count) of all the ports in a switch, somehow similar to the proposed *feedback comparison* probability management variant in Section 4.6.3. *Contention counters* introduced by Fuentes *et al.* [66] support non-minimal routing using traffic demand information, which is the output ports that would be used under minimal routing. Their *Explicit Contention Notification* (ECtN) mechanism routing algorithm based on those contention counters distributes explicit messages with port contention information, which is the traffic demand for each global port in a Dragonfly topology, resembling ECN. Unfortunately, unlike the proposal in Chapter 4, contention counters as employed by the authors are not amenable to a straightforward implementation in commodity switches.

The policies described in Section 4.6 replace QCN injection throttling at the NICs with adaptive routing in source switches, based on in-network congestion information sensed at

in-transit buffers. Endpoint congestion would still need to throttle injection, but as it has been already mentioned, this thesis is focused on in-network congestion and discards any evaluation of endpoint congestion. Injection throttling might be handled using the original QCN implementation or any other ECN mechanism, by identifying CNMs generated by endpoints and not intercepting them in the network switches. Alternative implementations may rely on congestion avoidance at the transport level, such as TCP congestion control, or on the use of proactive reservations to avoid saturating the network, such as SRP [94], CRP [134], SMSRP and LHRP [95].

Conditional flow rules based on OpenFlow were introduced in a different context in AVANT-GUARD by Shin *et al.* [169]. Their conditional flow rules are triggered when a potential attack is discovered to enforce the security policy. The proposed switch data-path is similar in both cases. However, the proposals introduced in Chapter 4 need to detect alternative triggers such as *pause* frames or an integer comparison between two numbers which ranges from 0 to 100: one random and another that acts as a probability. Moreover, NEC ProgrammableFlow [141] extended OpenFlow 1.0 to support conditional failover rules and conditional routing, but their implementation is used to mimic policy-based rather than adaptive routing. The work presented in Chapter 4 relied on multiple *class-of-service* levels with conditional OpenFlow rules to avoid deadlock. Alternative mechanisms rely on path restrictions, such as TCP-Bolt [174].

The power consumption of TCAM has been discussed in Section 4.3 and table size minimization has been considered across all Chapter 4. The power consumption devoted to signal transmission on links depends on the link technology, interconnect topology and link speed. Different topologies which target low overall power consumption have been considered across the aforementioned chapter. Congdon *et al.* [48] dissect the power consumption of an OpenFlow switch, but they do not consider the impact of the network topology. The impact of topology has been considered by Abts *et al.* [4] to dynamically reduce link speed to adapt to traffic load. However, they do not consider the impact of forwarding table organization. An implementation for HPC which relies on low-power Ethernet was presented in [161]. Similarly, ElasticTree [81] completely shuts down links and modifies routing to save link power.

7.2 Adapting the length of non-minimal paths

Several mechanisms have proposed different adaptions of *Valiant Load-Balancing* (VLB) routing algorithm, in which the intermediate router selection is not performed among *all* the routers in the network and then, the Valiant path is shortened. The original proposal for VLB in the Dragonfly by Kim *et al.* [108] selects a random intermediate *group*, instead of a *router*, $\text{VLB}_{lg\text{-}}$ policy following the terminology proposed by this dissertation. This reduces the length of the path and the amount of virtual channels but introduces pathological

performance issues under certain adversarial traffic patterns [72]. Similarly, the Dragonfly implementations proposed in Chapter 4 divert traffic using a single global hop for the phase A (VLB_{-g-}), which is equivalent to select a random *group* adjacent to the source switch; in this case, the implementation is constrained to using commodity Ethernet hardware without bookkeeping information in the packet headers. The *ACOR* routing proposed in Chapter 5 reduces the non-minimal path length, but falls back to the complete Valiant path in phase A under high injection loads to guarantee the absence of pathological issues. In *ACOR-Switch* implementation, routers maintain a given *ACOR level*, which indicates the specific path to use in VLB phase A. This level may increase or decrease, based on an indirect estimation of network congestion detected by a switch allocator retry. ACOR levels vary according to the sequence $-g- \leftrightarrow -gl \leftrightarrow lgl$. However, ACOR is oblivious to the traffic pattern present in the network because it takes the decision to route the packets minimally or non-minimally when it detects congestion on the output ports based on accounting the switch allocator retries. Hence, it requires that the congestion is propagated to the source for suffering allocator retries at source switch, leading to suboptimal results in certain cases that are overcome by the proposal in Chapter 6. T-UGAL by Rahman *et al.* [156] proposes a modified UGAL routing for Dragonfly networks with more than one link between groups (*global trunking*) which employs a subset of Valiant paths with shorter average path length. It reduces the number of hops in both phase A and B of the VLB path. However, it is based on an offline computation of the possible paths and, just like ACOR, it does not adapt the paths to the current traffic pattern. The proposal in Chapter 6, LIAN, overcomes this design limitation reacting to the traffic pattern inferred from the interconnection network through a set of counters in the routers.

Different proposals have implemented restricted and shorted variants of Valiant routing in other topologies. The proposal for *randomized routing in multidimensional square meshes* in [187] does not randomize all the N dimensions, but only $N - 1$; congestion is avoided assuming a single injector per switch, but this is not typically the case in current and forthcoming parallel systems. The *DAL* adaptive routing mechanism in HyperX [9] follows the idea of the *RVLB* technique, which is introduced in Section 5.2, misrouting only in the dimensions with offset. The Cray Cascade system [60] implements a mechanism similar to *restricted* Valiant technique, based on two different sets of tables [148]. Valiant routing in the indirect Orthogonal Fat-trees [185] and Multi-layer Full-Mesh [70] networks is restricted to switches with connected nodes in [103]. The already-mentioned proposal by Yébenes *et al.* [193] identifies the turn-around problem in Valiant routing in the SlimFly topology [28], and introduces a modified version of Valiant routing for this topology that only makes one non-minimal hop in phase A. However, they do not focus on other cases where path shortening improves performance and does not introduce congestion. Multiple short non-minimal paths are employed in other cases. For example, the Jellyfish random topology by Singla *et al.* [173] relies on a *k-shortest path* algorithm [194] to find a set of paths

(minimal and non-minimal) to each destination. However, it does not study path selection techniques, leaving that task to the end-to-end congestion control mechanism. Moreover, it would never consider maximum-length Valiant paths, which are required in some cases as observed in this dissertation. Furthermore, previous proposals do not formally prove the absence of pathological performance issues under any traffic pattern. ACOR and LIAN approaches, introduced in Section 5.4 and Section 6.2 respectively, can be adapted to such restricted routing variants and both fall back to the complete Valiant path in phase A under high load or congestion situations detected by the counters, which guarantees the absence of such hypothetical pathologies.

There is no awareness of any application of the re-computation of the R_{ROOT} based on network congestion to *VLB* as described in Section 5.3, but there are adaptive routing proposals that somehow resemble it. For example, the aforementioned *DAL* adaptive routing in HyperX [9] deroutes traffic in a given dimension based on the unavailability of output ports; therefore, it dynamically diverts traffic to an intermediate destination which is computed dynamically.

Previous works have considered the use of traffic counters to modify routing based on a central controller, such as Hedera [62]. They work on a per-flow granularity and their adaptation time is significantly larger than in LIAN. LIAN employs *extended* traffic counters to estimate *offered* traffic instead of *carried* traffic. In Hedera, this problem is dealt using an iterative algorithm. TPR [61] exploits *traffic counters* to modify UGAL threshold, but it fails to estimate offered traffic and does not leverage traffic information to shorten non-minimal paths.

8 Conclusions and Future Work

The evolution of computing systems resulting in faster, more capable and parallel systems makes the importance of interconnection networks be in constant growth. They play a crucial role on determining the overall performance and cost of the computing systems. The interconnection network topology determines the performance bounds of the network. All the capabilities offered by the topology must be exploited by the routing algorithm to accomplish as much as possible of these performance limits.

The PhD work presented in this dissertation started with the objective of designing non-minimal adaptive routing algorithms for efficient interconnection networks. This is approached from two orthogonal points of view: 1) decoupling the selection between minimal and non-minimal paths from flow-control credits and enabling the use of commodity Ethernet technology for scalable exascale environments; and 2) employing non-minimal paths with different path lengths based on the network conditions which allows to optimize the latency to the current situation of the network.

Through the completion of the PhD work presented in this dissertation, both objectives have been achieved as it is further detailed in the following section, in which the conclusions of this thesis are explained. The result of this PhD thesis is a set of proposals which allows to implement a non-minimal adaptive routing algorithm for efficient interconnection networks because: 1) the *QCN-Switch* adaptive routing algorithm, which resembles the performance of state-of-the-art custom high-performance computing proposals, implemented upon the introduced architecture allows to deploy scalable HPC interconnection networks over commodity Ethernet OpenFlow network devices; and 2) *ACOR* and *LIAN* allow the optimization of latency by adapting the length of non-minimal paths to the current network conditions.

Chapter contents

8.1 Conclusions

This dissertation explains different proposals to design and implement source-adaptive non-minimal routing algorithms for efficient interconnection networks. The adaptability of these routing algorithms, based on the network conditions, allows them to select between minimal and non-minimal paths for each packet and to define non-minimal paths with different lengths. In summary, this dissertation proposes four source adaptive non-minimal routing algorithms and switch architecture changes to be capable of implementing a pro-active and per-packet basis adaptive routing depending on the network conditions.

The first proposed routing algorithm, *MAR-bP* is initially proposed to overcome the lack of flow-control credits on commodity Ethernet networks. Its implementation is naïve, since it defines a fixed non-minimal path based in solely one hop. With the objective of overcoming the limitations of the previous proposal, *QCN-Switch* is introduced extending the architecture introduced with MAR-bP. It enables the possibility of selecting the output port statistically based on explicit congestion notification messages snooped from the network. Although the QCN-Switch implementation is also naïve because it is initially defined to use non-minimal paths relying on one hop[1], this architecture together with the implemented routing offers the possibility to select between minimal and non-minimal paths packet-by-packet over a novel practicable implementation of HPC interconnection network built upon commodity Ethernet OpenFlow devices.

As a first approach to the second sub-objective, two techniques (*RVLB* and *VLB-Recomp*) regarding the selection of the *Valiant Load-Balancing* (VLB) intermediate router (R_{ROOT}) are introduced to develop *ACOR*, a source-adaptive routing algorithm capable of adapting the number of hops in the non-minimal paths based on live network conditions. This routing proposal is topology-agnostic and employs the amount of routing attempts done by packets to take the misrouting decision. Focused on the same sub-objective, *LIAN* routing is proposed. While, same as ACOR, it aims to optimize the amount of hops in non-minimal paths based on network conditions and uses RVLB mechanism, it is based on a series of network counters and is topology-dependent. Note that these two latest proposals cannot be directly implemented over the network architecture proposed in Chapter 4 due to technology constraints. However, the idea of both proposals is compatible with that architecture and could be combined if technology for programming the routers' data-path is used.

Chapter 4 introduces a novel practicable implementation of *High-Performance Computing* (HPC) interconnection networks based on commodity Ethernet switches relying on a hierarchical routing and addressing based on *location-dependent MAC addresses*, *TCAM rules compaction*, a *dynamic mechanism to assign location-dependent MAC addresses to terminals* and *pro-active conditional OpenFlow rules* for adaptive routing. The implementation of the

[1]The proposed implementation employs one hop for the non-minimal paths but this could be enlarged employing more forwarding rules than those initially proposed.

above-mentioned set of mechanisms is realistic and requires minimal changes in OpenFlow switches allowing the implementation of low-power and low-diameter networks based on commodity Ethernet.

Whereas most adaptive routing mechanisms proposed for low-diameter networks rely on local congestion information, such as credits, to decide between using minimal and non-minimal paths to forward each new packet, due to technology constraints, this routing sets up its adaptability on *explicit congestion notifications*. The proposed *QCN-Switch* uses statistical routing driven by intercepting explicit congestion notification messages generated using commodity QCN[2] *congestion points* in switches.

The steady-state performance of the final *QCN-Switch* design relying on output-port sampling and *feedback comparison* probability management variant is comparable to state-of-the-art sophisticated HPC routing alternatives such as Piggyback. Considering transient changes of traffic, while the resulting design does not adapt routing as quickly as other mechanisms that rely on credits, it responds in a sub-millisecond time frame, which is typically enough for most applications. A sensitivity analysis presents the trade-offs of the design, particularly improving performance for uniform or adversarial traffic patterns. In conclusion, it is proposed a naïve one non-minimal hop[1] adaptive routing which leverages ECN notifications to send each packet by minimal or non-minimal path and it is provided a feasible switch design for low-diameter networks based on commodity white-box Ethernet switches.

Valiant load-balancing routing algorithm used to achieve good performance under adversarial traffic patterns in low-diameter networks, such as Dragonfly, extends the length of the path and so, increases the base network latency. Two improvements to VLB are introduced in Chapter 5: 1) *RVLB* improves performance for traffic with locality, selecting the intermediate router (R_{ROOT}) in the same network partition as the source and destination; and 2) *RVLB-Recomp* avoids *head-of-line blocking* at injection by selecting an alternative R_{ROOT} when the output port associated with the current is stalled.

Based on these mechanisms, *ACOR: Adaptive Congestion-Oblivious Routing* is also introduced in that chapter. The goal of ACOR is to optimize the common case exhibiting minimal latency, while supporting pathological traffic patterns with longer paths. It applies the two above-mentioned improvements to Valiant by extending the *restrictions* which determine the path of VLB phase A to global traffic and expanding the idea of path *recomputation* to adapt to network conditions, changing the non-minimal path following a given sequence ordered by path length. It prevents variability in the results through a simple hysteresis mechanism and the implementation is relatively simple. However, it cannot be implemented straightforward over the router architecture presented in Chapter 4 due to technology constraints.

Same as VLB, ACOR does not send traffic minimally, so its performance under benign traffic is suboptimal. For this reason, the ACOR mechanism has been coupled with a source-

[2] *Quantized Congestion Notification* (QCN, [89]).

adaptive non-minimal routing algorithm to decide between minimal or ACOR path for each injected packet. In this case, Piggyback has been employed but the switch architecture presented in Chapter 4 could be employed if data-path programmable routers are available. The combination of PB and ACOR, hereon *PB-ACOR*, selects the shortest feasible non-minimal path, but only when the minimal route is congested. This mechanism maintains the benefits of ACOR regarding adversarial traffic and is competitive under benign traffic. Evaluation results show that all the ACOR variants avoid any throughput pathologies and unfairness issues and reduce base latency by up to 28% compared to a Valiant with *restricted* and *recomputation* techniques already applied (*RVLB-Recomp*). Furthermore, PB-ACOR reaches high throughput and optimal latency under uniform traffic pattern and achieves rivaling throughput and significantly lower latency compared to base PB.

Following the same trend of ACOR to reduce the number of hops in the non-minimal paths, Chapter 6 introduces *LIAN: Latency-Improved Adaptive Non-minimal routing for Dragonfly networks*. This proposal pursues the idea of shortening the length of non-minimal paths based on *offered load* estimation. The counters proposed resembles the performance counters already present in modern routers. Based on *global* counters which tracks the offered load to each group in the network some *intermediate-local* counters are derived by a simple calculation combining some of them. The equation that determines how to combine these global counters is extrapolated from the symmetry exposed by the *palmtree* regular global link arrangement of the Dragonfly topology.

Both, global and intermediate-local counters are used to statistically infer the congestion on intermediate groups and analyze the demand of global output ports. Based on that, LIAN is able to determine the *VLB phase A policy* that packets must employ, if they are sent non-minimally, to dynamically optimize the length of non-minimal paths. LIAN overcomes the need of ACOR to analyze the congestion at the source because it is based on inferring the traffic load rather than the *recomputation* technique. LIAN also modulates the UGAL threshold T based on global counters to detect the traffic offered load and a small series of empirically obtained thresholds which establishes three intensity zones.

The results show that introduced *extended* counters correctly identify network traffic in all possible ranges, leading to accurate routing under both benign and adversarial traffic patterns. Also, LIAN routing achieves a performance equal to or better than the best oblivious routing for each traffic pattern analyzed. In those conditions, LIAN also achieves optimal latency selecting non-minimal paths with the ideal number of hops that should be used when VLB is required to avoid congestion. This feature allows latency reductions up to 30.0% over PB-ACOR. Furthermore, its throughput is very close to the theoretical maximum for each traffic pattern analyzed, which makes it very competitive compared with the state-of-the-art adaptive routing algorithms. Moreover, LIAN does not present obvious effects of routing unfairness and its throughput is stable after saturation point. In addition, performance results exhibit an almost-immediate reaction time to traffic changes, both changing from one

traffic pattern to another and changing the offered load for the same traffic pattern. In conclusion, LIAN is the first source-adaptive non-minimal routing algorithm that combines the desirable properties of: 1) relying solely on local information; 2) employing the shortest VLB phase A path allowed by the live network situation without introducing congestion; and 3) providing stable saturation throughput.

Section 1.2 sets the objective of designing non-minimal adaptive routing algorithms for efficient interconnection networks and splits it in the following two orthogonal sub-objectives: 1) design a routing capable of selecting between minimal or non-minimal path without relying on credits, and 2) design a routing capable of adapting the number of hops in the non-minimal paths to the network conditions instead of directly using paths according to Valiant load-balancing algorithm. In light of the above conclusions, the two sub-objectives have been addressed by this work because; 1) Chapter 4 addresses the first sub-objective providing *QCN-Switch*, which is a routing algorithm that decides if routing each packet minimally or non-minimally based on ECN messages snooped from the network, and a novel practicable implementation of HPC networks based on commodity Ethernet switches. 2) Chapter 5 and Chapter 6 provide two different routing algorithms to optimize the number of hops in the non-minimal paths. Moreover, an adaptive selection between minimal and non-minimal paths is also done by these routings, however this is based on UGAL and flow-control credits. For this reason, a combination of LIAN plus QCN-Switch is a straightforward future work as it is explained later.

The result of this PhD thesis is a set of proposals which allows to implement a non-minimal adaptive routing algorithm for efficient interconnection networks contributing separately to two orthogonal aspects of it. On the one hand, the *QCN-Switch* implemented upon the presented architecture allows to deploy scalable HPC interconnection networks over commodity Ethernet OpenFlow network devices employing an adaptive routing which resembles the performance of state-of-the-art custom HPC proposals. On the other hand, *ACOR* allows to make a topology-agnostic traffic engineering which adapts the packet latency to the current network conditions. Additionally, *LIAN* improves the previous latency reduction and optimizes the length of non-minimal paths in a Dragonfly topology using the palmtree global link arrangement.

8.2 Future directions

This dissertation has accomplished the objective from which this PhD work started, and to conduct that, several contributions to the study of non-minimal adaptive routings for efficient interconnection networks have been done. Nevertheless, each proposal literally opens the door to new challenges that can be grouped as new research lines. A few interesting ones, which can be built upon this work, are briefly introduced next.

- *Implement and develop* LIAN *together with* QCN-Switch *over programmable network devices:* Implement LIAN over QCN-Switch using commodity Ethernet with programmable data-path network devices to provide an optimized solution for a scalable Dragonfly network built upon commodity Ethernet devices and focused on low-power and on optimizing the average latency suffered by packets.

- *Identify and handle endpoint and network congestion separately:* In a case of endpoint congestion, exploiting path diversity through adaptive non-minimal routing algorithms increases the pressure on the interconnection network and can reduce the performance instead of increasing it. Detecting this situation combined with any type of injection throttling can prevent the spread of the congestion to the whole network and the performance degradation.

- *Cooperation between terminals and routers:* Computational power of source hosts are bigger than routers and also, terminals know partially the following packets to inject to the network. However, the router has the routing tables and knows the status of the neighbors and network links. A bit of cooperation between terminals and routers by sharing information may improve the routing decisions on source adaptive routing algorithms.

- *Translate* restricted *technique for Valiant load-balancing routing algorithm to other topologies:* The locality of Dragonfly or Flattened Butterfly topologies is quite direct, based on groups or dimensions, but how to apply RVLB to other topologies without suffering congestion under any traffic pattern is not trivial.

- *Translate the topology-dependent counters from* LIAN *to other topologies:* The rotational symmetry exploited in Chapter 6 to optimize the routing based in *intermediate-local* counters is not directly translatable to other topologies or other Dragonfly global link arrangements. However, it could be an interesting study.

- *Implement and develop* ACOR *over programmable data-path OpenFlow network devices:* The advent of white-box data-path programmable network devices could allow the implementation of the *recomputation* technique and then, the ACOR routing algorithm can be implemented over commodity Ethernet with programmable data-path OpenFlow network devices. The selection of non-minimal path based on ACOR can be combined with Piggyback or UGAL routing algorithms based on credits as well as over the architecture for HPC systems based on ECN messages like Ethernet 802.1Qau.

- *Employ the* LIAN counters *also to decide between minimal or non-minimal routing for each packet:* Traffic counters employed to modulate UGAL threshold could be used

directly to determine the injection of the packets following minimal or non-minimal paths omitting the use of UGAL and credits.

- *Extend* LIAN *to be able to determine the length of the whole non-minimal path:* LIAN routing algorithm optimizes the length of the VLB phase A to the detected network conditions but does not analyze the second minimal phase of VLB routing algorithm. The selection of intermediate router R_{ROOT} can be *restricted* to determine the whole non-minimal path on the source router.

8.3 Publications

The research presented in this PhD thesis is partially supported or related with the following publications obtained during the training period to accomplish the PhD:

- M. Benito, E. Vallejo, and R. Beivide, "LIAN: Latency-Improved Adaptive Non-minimal routing for Dragonfly Networks," Submitted and pending review.
- M. Benito, P. Fuentes, E. Vallejo, and R. Beivide, "ACOR: Adaptive congestion-oblivious routing in Dragonfly networks," *Journal of Parallel and Distributed Computing*, vol. 131, pp. 173–188, Sep. 2019. DOI: 10.1016/j.jpdc.2019.04.022
- ——, "Analysis and improvement of Valiant routing in low-diameter networks," in *2018 IEEE 4th International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, IEEE, Feb. 2018. DOI: 10.1109/hipineb.2018.00009
- M. Benito, E. Vallejo, C. Izu, and R. Beivide, "Non-minimal adaptive routing based on explicit congestion notifications," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 2, e4440, Jan. 2018. DOI: 10.1002/cpe.4440
- M. Benito, E. Vallejo, R. Beivide, and C. Izu, "Extending commodity OpenFlow switches for large-scale HPC deployments," in *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, IEEE, Feb. 2017. DOI: 10.1109/hipineb.2017.12
- N. Rajovic, A. Rico, F. Mantovani, D. Ruiz, J. O. Vilarrubi, C. Gomez, L. Backes, D. Nieto, H. Servat, X. Martorell, J. Labarta, E. Ayguade, C. Adeniyi-Jones, S. Derradji, H. Gloaguen, P. Lanucara, N. Sanna, J.-F. Mehaut, K. Pouget, B. Videau, E. Boyer, M. Allalen, A. Auweter, D. Brayford, D. Tafani, V. Weinberg, D. Brommel, R. Halver, J. H. Meinke, R. Beivide, M. Benito, E. Vallejo, M. Valero, and A. Ramirez, "The Mont-Blanc prototype: An alternative approach for HPC systems," in *SC'16: International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, Nov. 2016. DOI: 10.1109/sc.2016.37

- M. Benito, E. Vallejo, and R. Beivide, "On the use of commodity Ethernet technology in exascale HPC systems," in *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*, IEEE, Dec. 2015. DOI: 10.1109/hipc.2015.32

The following list of publications are out of the scope of this dissertation but were performed during the same period:

- P. Fuentes, M. Benito, E. Vallejo, J. L. Bosque, R. Beivide, A. Anghel, G. Rodríguez, M. Gusat, C. Minkenberg, and M. Valero, "A scalable synthetic traffic model of Graph500 for computer networks analysis," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 24, e4231, Jul. 2017. DOI: 10.1002/cpe.4231

- E. Vallejo, P. Fuentes, and M. Benito, "Aprendizaje autónomo del estudiante apoyado en recursos audiovisuales en el contexto de un grado de ingeniería informática: Experiencias con metodologías de enseñanza activas," in *Libro de Actas IN-RED 2017 - III Congreso Nacional de Innovación Educativa y de Docencia en Red*, Universitat Politècnica València, Jul. 2017. DOI: 10.4995/inred2017.2017.6753

Bibliography

[1] *802.1D-1990: IEEE standard for local and metropolitan area networks: Media access control (MAC) bridges*, 3 Park Avenue, New York, Usa: IEEE Computer Society, 1991. DOI: 10.1109/IEEESTD.1991.101050 (see p. 64)

[2] F. Abel, C. Minkenberg, R. Luijten, M. Gusat, and I. Iliadis, "A four-terabit packet switch supporting long round-trip times," *IEEE Micro*, vol. 23, no. 1, pp. 10–24, Jan. 2003. DOI: 10.1109/mm.2003.1179894 (see p. 14)

[3] D. Abts, A. Bataineh, S. Scott, G. Faanes, J. Schwarzmeier, E. Lundberg, T. Johnson, M. Bye, and G. Schwoerer, "The Cray BlackWidow," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing - SC'07*, ACM Press, 2007. DOI: 10.1145/1362622.1362646 (see pp. 13, 21)

[4] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," in *International Symposium on Computer Architecture (ISCA)*, Saint-Malo, France: ACM, 2010, pp. 338–347, ISBN: 978-1-4503-0053-7. DOI: 10.1145/1815961.1816004 (see pp. 84, 160)

[5] A. Agarwal, "Limits on interconnection network performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398–412, 1991. DOI: 10.1109/71.97897 (see pp. 13, 21)

[6] B. Agrawal and T. Sherwood, "Ternary CAM power and delay model: Extensions and uses," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 5, pp. 554–564, 2008, ISSN: 1063-8210. DOI: 10.1109/TVLSI.2008.917538 (see p. 82)

[7] B. Agrawal and T. Sherwood, *Ternary CAM (TCAM) power and delay modeling*, accessed: 2015-02-09, Mar. 2006. [Online]. Available: https://sites.cs.ucsb.edu/~arch/mem-model (see p. 82)

[8] V. Ahlgren, S. Andersson, J. Brandt, N. Cardo, S. Chunduri, J. Enos, P. Fields, A. Gentile, R. Gerber, M. Gienger, J. Greenseid, A. Greiner, B. Hadri, Y. He, D. Hoppe, U. Kaila, K. Kelly, M. Klein, A. Kristiansen, S. Leak, M. Mason, K. Pedretti, J.-G. Piccinali, J. Repik, J. Rogers, S. Salminen, M. Showerman, C. Whitney, and J. Williams, "Large-scale system monitoring experiences and recommendations," in *2018 IEEE*

International Conference on Cluster Computing (CLUSTER), IEEE, Sep. 2018. DOI: 10.1109/cluster.2018.00069 (see p. 131)

[9] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "HyperX: Topology, routing, and packaging of efficient large-scale networks," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 1–11 (see pp. 21, 158, 161, 162)

[10] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar, and M. Seaman, "Data center transport mechanisms: Congestion control theory and IEEE standardization," in *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, Sep. 2008, pp. 1270–1277 (see pp. 82, 83)

[11] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *ACM SIGCOMM Conference*, 2010, pp. 63–74, ISBN: 978-1-4503-0201-2 (see p. 158)

[12] G. Almasi, *Highly parallel computing*. Redwood City, CA: Benjamin - Cummings Publishing Co. Inc., 1989, ISBN: 9780805301779 (see pp. i, 1)

[13] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, *Cray XC series network, WP-Aries01-1112*, accessed: 2015-06-10, 2012. [Online]. Available: https://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf (see pp. 22, 108)

[14] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High-speed switch scheduling for local-area networks," *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319–352, Nov. 1993. DOI: 10.1145/161541.161736 (see p. 15)

[15] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, N. Ni, and R. Rajamony, "The PERCS high-performance interconnect," in *2010 18th IEEE Symposium on High Performance Interconnects*, IEEE, Aug. 2010. DOI: 10.1109/hoti.2010.16 (see pp. 22, 109, 159)

[16] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The NAS parallel benchmarks, summary and preliminary results," in *ACM/IEEE Conference on Supercomputing*, ser. Supercomputing'91, Albuquerque, New Mexico, United States: ACM, 1991, pp. 158–165, ISBN: 0-89791-459-7. DOI: 10.1145/125826.125925 (see p. 62)

[17] E. Baubold and J. D. Haenle, "A method of deadlock-free resource allocation and flow control in packet networks," in *Proceedings of the Third International Conference on Computer Communication, Toronto, Canada, August 3-6, 1976*, P. K. Verma, Ed., International Council for Computer Communication, 1976, pp. 483–487 (see p. 16)

[18] D. U. Becker, "Efficient microarchitecture for network-on-chip routers," accessed: 2016-02-17, Ph.D. dissertation, Stanford University, Aug. 2012. [Online]. Available: http://purl.stanford.edu/wr368td5072 (see p. 17)

[19] D. U. Becker and W. J. Dally, "Allocator implementations for network-on-chip routers," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis - SC'09*, ACM Press, 2009. DOI: 10.1145/1654059.1654112 (see p. 17)

[20] M. Belka, M. Doubet, S. Meyers, R. Momoh, D. Rincon-Cruz, and D. P. Bunde, "New link arrangements for dragonfly networks," in *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, IEEE, Feb. 2017. DOI: 10.1109/hipineb.2017.14 (see pp. 24, 25)

[21] M. Benito, P. Fuentes, E. Vallejo, and R. Beivide, "Analysis and improvement of Valiant routing in low-diameter networks," in *2018 IEEE 4th International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, IEEE, Feb. 2018. DOI: 10.1109/hipineb.2018.00009 (see pp. iv, vi, 114, 116, 118)

[22] ——, "ACOR: Adaptive congestion-oblivious routing in Dragonfly networks," *Journal of Parallel and Distributed Computing*, vol. 131, pp. 173–188, Sep. 2019. DOI: 10.1016/j.jpdc.2019.04.022 (see pp. iv, vi, 125)

[23] M. Benito, E. Vallejo, and R. Beivide, "On the use of commodity Ethernet technology in exascale HPC systems," in *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*, IEEE, Dec. 2015. DOI: 10.1109/hipc.2015.32 (see pp. iv, v)

[24] M. Benito, E. Vallejo, R. Beivide, and C. Izu, "Extending commodity OpenFlow switches for large-scale HPC deployments," in *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, IEEE, Feb. 2017. DOI: 10.1109/hipineb.2017.12 (see pp. iv, v, 83)

[25] M. Benito, E. Vallejo, C. Izu, and R. Beivide, "Non-minimal adaptive routing based on explicit congestion notifications," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 2, e4440, Jan. 2018. DOI: 10.1002/cpe.4440 (see pp. iv, v, 89)

[26] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *10th ACM SIGCOMM Conference on Internet Measurement (IMC'10)*, Melbourne, Australia, 2010, pp. 267–280, ISBN: 978-1-4503-0483-2. DOI: 10.1145/1879141.1879175 (see p. 63)

[27] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '11, Tokyo, Japan: ACM, 2011, 8:1–8:12, ISBN: 978-1-4503-1041-3. DOI: 10.1145/2079296.2079304 (see p. 158)

[28] M. Besta and T. Hoefler, "SlimFly: A cost effective low-diameter network topology," in *IEEE/ACM Intl. Conf. on High Performance Computing, Networking, Storage and Analysis (SC'14)*, New Orleans, LA, USA, 2014 (see pp. 21, 39, 161)

[29] A. Bhatele, N. Jain, Y. Livnat, V. Pascucci, and P.-T. Bremer, "Analyzing network health and congestion in Dragonfly-based supercomputers," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, May 2016. DOI: 10.1109/ipdps.2016.123 (see p. 53)

[30] Bhuyan and Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Transactions on Computers*, vol. C-33, no. 4, pp. 323–333, Apr. 1984. DOI: 10.1109/tc.1984.1676437 (see p. 21)

[31] A. Bicas, "IBM power system AC922 introduction and technical overview," IBM Corp., Tech. Rep., Mar. 2018, REDP-5472-00, accessed: 2020-08-13 (see p. 61)

[32] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, and R. C. Zak, "Enabling scalable high-performance systems with the Intel Omni-Path architecture," *IEEE Micro*, vol. 36, no. 4, pp. 38–47, 2016, ISSN: 0272-1732 (see pp. 7, 14, 157, 158)

[33] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W.-K. Su, "Myrinet: A gigabit-per-second local area network," *IEEE Micro*, vol. 15, no. 1, pp. 29–36, 1995. DOI: 10.1109/40.342015 (see p. 28)

[34] P. Bosshart, G. Varghese, D. Walker, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, and A. Vahdat, "P4," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, Jul. 2014. DOI: 10.1145/2656877.2656890 (see p. 42)

[35] J. M. Brandt, E. Froese, A. C. Gentile, L. Kaplan, B. A. Allan, and E. J. Walsh, "Network performance counter monitoring and analysis on the Cray XC platform," May 2016 (see p. 131)

[36] R. Brightwell, K. Pedretti, K. Underwood, and T. Hudson, "SeaStar interconnect: Balanced bandwidth for scalable performance," *IEEE Micro*, vol. 26, no. 3, pp. 41–57, May 2006. DOI: 10.1109/mm.2006.65 (see p. 21)

[37] "Brocade VCS fabric technical architecture," Brocade Communications Systems, Inc, Tech. Rep., 2012 (see p. 157)

[38] C. Camarero, E. Vallejo, and R. Beivide, "Topological characterization of Hamming and Dragonfly networks and its implications on routing," *ACM Transactions on Architecture and Code Optimization*, vol. 11, no. 4, pp. 1–25, Dec. 2014. DOI: 10.1145/2677038 (see pp. ii, 23–25)

[39] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM'07*, ACM Press, 2007. DOI: 10.1145/1282380.1282382 (see p. 41)

[40] B. Casemore, L. Rosenberg, R. Brothers, R. Costello, R. Mehra, P. Jirovsky, and N. Greene, *Worldwide enterprise communications and datacenter networks 2014: Top 10 predictions*, IDC report, 2014 (see p. 61)

[41] K.-Y. K. Chang, S.-T. Chuang, N. McKeown, and M. Horowitz, "A 50 Gb/s 32×32 CMOS crossbar chip using asymmetric serial links," in *1999 Symposium on VLSI Circuits. Digest of Papers (IEEE Cat. No.99CH36326)*, Japan Society Applied Physics (JSAP), 1999. DOI: 10.1109/vlsic.1999.797221 (see p. 14)

[42] F. M. Chiussi, J. G. Kneuer, and V. P. Kumar, "Low-cost scalable switching solutions for broadband networking: The ATLANTA architecture and chipset," *IEEE Communications Magazine*, vol. 35, no. 12, pp. 44–53, Dec. 1997. DOI: 10.1109/mcom.1997.642833 (see p. 14)

[43] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input output queued switch," in *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, IEEE, 1999. DOI: 10.1109/infcom.1999.751673 (see p. 15)

[44] Cisco Inc., *Cisco Catalyst 3750-X and 3560-X series switches data sheet*, accessed: 2015-01-28, May 2013. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-3560-x-series-switches/data_sheet_c78-584733.html (see p. 64)

[45] ——, "Nexus 7000 FabricPath whitepaper version 2.0," Cisco, Tech. Rep., Sep. 2013 (see pp. 64, 157)

[46] ——, *Cisco Nexus 3548-X and 3524-X switches data sheet*, accessed: 2015-01-28, 2015. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-3548-switch/data_sheet_c78-707001.html (see p. 64)

[47] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, Mar. 1953. DOI: 10.1002/j.1538-7305.1953.tb01433.x (see pp. 5, 21)

[48] P. Congdon, P. Mohapatra, M. Farrens, and V. Akella, "Simultaneously reducing latency and power consumption in OpenFlow switches," *Networking, IEEE/ACM Transactions on*, vol. 22, no. 3, pp. 1007–1020, Jun. 2014, ISSN: 1063-6692. DOI: 10.1109/TNET.2013.2270436 (see pp. 82, 84, 160)

[49] Cray Inc., "About Aries hardware counters," Tech. Rep., Nov. 2018, accessed: 2020-07-30 and downloaded as PDF with publication number S-0045-40. [Online]. Available: https://pubs.cray.com/bundle/Aries_Hardware_Counters_S-0045-40/page/About_Aries_Hardware_Counter_S-0045.html (see pp. vii, 108, 131)

[50] Dally and Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547–553, May 1987. DOI: 10.1109/tc.1987.1676939 (see p. 34)

[51] W. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 775–785, Jun. 1990. DOI: 10.1109/12.53599 (see pp. 13, 21)

[52] ——, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, Mar. 1992. DOI: 10.1109/71.127260 (see pp. 34, 39)

[53] W. J. Dally and C. L. Seitz, "The torus routing chip," *Distributed Computing*, vol. 1, no. 4, pp. 187–196, Dec. 1986. DOI: 10.1007/bf01660031 (see p. 34)

[54] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004, ISBN: 9780080497808 (see pp. i, iv, 3, 8, 16–19, 21, 27, 34, 35, 38, 47, 50, 51, 53)

[55] A. Daryin and A. Korzh, "Early evaluation of direct large-scale InfiniBand networks with adaptive routing," *Supercomputing frontiers and innovations*, vol. 1, no. 3, pp. 56–69, 2015, ISSN: 2313-8734 (see p. 157)

[56] A. DeConinck, H. A. Nam, D. Morton, A. Bonnie, C. Lueninghoener, J. M. Brandt, A. C. Gentile, K. Pedretti, A. M. Agelastos, C. T. Vaughan, S. D. Hammond, B. A. Allan, M. Davis, and J. J. Repik, "Runtime collection and analysis of system metrics for production monitoring of Trinity Phase II," May 2017 (see p. 131)

[57] S. Derradji, T. Palfer-Sollier, J.-P. Panziera, A. Poudes, and F. W. Atos, "The BXI interconnect architecture," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, IEEE, Aug. 2015. DOI: 10.1109/hoti.2015.15 (see p. 14)

[58] J. Domke, S. Matsuoka, I. Radanov, Y. Tsushima, T. Yuki, A. Nomura, S. Miura, N. McDonald, D. L. Floyd, and N. Dube, "The first supercomputer with HyperX topology: A viable alternative to fat-trees?" In *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*, IEEE, Aug. 2019. DOI: 10.1109/hoti.2019.00013 (see p. 22)

[59] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*, M. K. Publishers, Ed. Elsevier Science & Technology, Jul. 29, 2002, 624 pp., ISBN: 978-1558608528 (see pp. i, ii, 3, 40)

[60] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard, "Cray Cascade: A scalable HPC system based on a Dragonfly network," in *2012 International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, Nov. 2012. DOI: 10.1109/sc.2012.39 (see pp. 22, 29, 39, 109, 159, 161)

[61] P. Faizian, J. F. Alfaro, M. S. Rahman, M. A. Mollah, X. Yuan, S. Pakin, and M. Lang, "TPR: Traffic pattern-based adaptive routing for dragonfly networks," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 4, pp. 931–943, Oct. 2018, ISSN: 2372-207X. DOI: 10.1109/TMSCS.2018.2877264 (see pp. 131, 142, 162)

[62] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2010 (see pp. 62, 76, 158, 162)

[63] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *ACM SIGCOMM Conference*, New Delhi, India, 2010, pp. 339–350, ISBN: 978-1-4503-0201-2 (see pp. 62, 76)

[64] M. Flajslik, E. Borch, and M. A. Parker, "Megafly: A topology for exascale systems," in *Lecture Notes in Computer Science*, Springer International Publishing, 2018, pp. 289–310. DOI: 10.1007/978-3-319-92040-5_15 (see pp. 21, 29)

[65] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, Aug. 1993, ISSN: 1063-6692 (see p. 158)

[66] P. Fuentes, E. Vallejo, M. Garcia, R. Beivide, G. Rodriguez, C. Minkenberg, and M. Valero, "Contention-based nonminimal adaptive routing in high-radix networks," in *2015 IEEE International Parallel and Distributed Processing Symposium*, 2015, pp. 103–112 (see p. 159)

[67] P. Fuentes, M. Benito, E. Vallejo, J. L. Bosque, R. Beivide, A. Anghel, G. Rodríguez, M. Gusat, C. Minkenberg, and M. Valero, "A scalable synthetic traffic model of Graph500 for computer networks analysis," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 24, e4231, Jul. 2017. DOI: 10.1002/cpe.4231 (see p. iv)

[68] P. Fuentes, E. Vallejo, C. Camarero, R. Beivide, and M. Valero, "Network unfairness in Dragonfly topologies," *The Journal of Supercomputing*, vol. 72, no. 12, pp. 4468–4496, 2016, ISSN: 1573-0484. DOI: 10.1007/s11227-016-1758-z (see pp. 30, 149)

[69] Fujitsu, *Supercomputer Fugaku: The world's top-level supercomputer*, accessed: 2020-08-13. [Online]. Available: https://www.fujitsu.com/global/about/innovation/fugaku (see p. 61)

[70] Fujitsu Laboratories Ltd., *Fujitsu laboratories develops technology to reduce network switches in cluster supercomputers by 40%, maintains network performance, lowers energy consumption*, accessed: 2015-07-14, Jul. 2014. [Online]. Available: https://www.fujitsu.com/global/about/resources/news/press-releases/2014/0715-02.html (see p. 161)

[71] Fujitsu Limited, "Fujitsu processor A64FX," Fujitsu, Tech. Rep., 2020, accessed: 2020-08-13. [Online]. Available: https://www.fujitsu.com/downloads/SUPER/a64fx/a64fx_datasheet.pdf (see p. 61)

[72] M. Garcia, E. Vallejo, R. Beivide, M. Odriozola, C. Camarero, M. Valero, G. Rodriguez, J. Labarta, and C. Minkenberg, "On-the-fly adaptive routing in high-radix hierarchical networks," in *2012 41st International Conference on Parallel Processing*, IEEE, Sep. 2012. DOI: 10.1109/icpp.2012.46 (see pp. 25, 30, 161)

[73] M. García, P. Fuentes, M. Benito, M. Odriozola, E. Vallejo, and R. Beivide, *FOGSim interconnection network simulator*, accessed: 2020-08-20, 2014. [Online]. Available: https://www.atc.unican.es/sw_fogsim.html (see pp. v, 51)

[74] M. García, E. Vallejo, R. Beivide, M. Odriozola, and M. Valero, "Efficient routing mechanisms for dragonfly networks," in *The 42nd International Conference on Parallel Processing (ICPP-42)*, 2013 (see pp. 61, 76, 158, 159)

[75] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F. Naven, "Dynamic evolution of congestion trees: Analysis and impact on switch architecture," in *High Performance Embedded Architectures and Compilers: First International Conference, HiPEAC 2005, Barcelona, Spain, November 17-18, 2005. Proceedings*, T. Conte, N. Navarro, W.-m. W. Hwu, M. Valero, and T. Ungerer, Eds. Springer Berlin Heidelberg, 2005, pp. 266–285, ISBN: 978-3-540-32272-6 (see p. 80)

[76] M. Gerla and L. Kleinrock, "Congestion control in interconnected LANs," *IEEE Network*, vol. 2, no. 1, pp. 72–76, Jan. 1988. DOI: 10.1109/65.3241 (see p. 35)

[77] B. Goglin, "High-performance message-passing over generic Ethernet hardware with Open-MX," *Parallel Computing*, vol. 37, no. 2, pp. 85–100, 2011, ISSN: 0167-8191 (see p. 64)

[78] P. Gratz, B. Grot, and S. W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, IEEE, Feb. 2008. DOI: 10.1109/hpca.2008.4658640 (see p. 27)

[79] K. Günther, "Prevention of deadlocks in packet-switched data transport systems," *IEEE Transactions on Communications*, vol. 29, no. 4, pp. 512–524, Apr. 1981. DOI: 10.1109/tcom.1981.1095021 (see p. 39)

[80] E. Hastings, D. Rincon-Cruz, M. Spehlmann, S. Meyers, A. Xu, D. P. Bunde, and V. J. Leung, "Comparing global link arrangements for Dragonfly networks," in *2015 IEEE International Conference on Cluster Computing*, IEEE, Sep. 2015. DOI: 10.1109/cluster.2015.57 (see pp. 24, 25)

[81] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving energy in data center networks," in *7th USENIX Conference on Networked Systems Design and Implementation (NSDI'10)*, San Jose, California: USENIX Association, 2010, pp. 17–17 (see pp. 158, 160)

[82] A. J. Hoffman and R. R. Singleton, "On Moore graphs with diameters 2 and 3," *IBM Journal of Research and Development*, vol. 4, no. 5, pp. 497–504, Nov. 1960. DOI: 10.1147/rd.45.0497 (see pp. iv, 8)

[83] C. Hopps, "Analysis of an equal-cost multi-path algorithm," The Internet Society, Tech. Rep., Nov. 2000. DOI: 10.17487/rfc2992 (see pp. 72, 159)

[84] I. Y.-L. Hsiao, D.-H. Wang, and C.-W. Jen, "Power modeling and low-power design of content addressable memories," in *IEEE Intl. Symp. on Circuits and Systems (ISCAS)*, vol. 4, 2001, 926–929 vol. 4. DOI: 10.1109/ISCAS.2001.922390 (see p. 82)

[85] IBM, "Overview of the IBM Blue Gene/P project," *IBM Journal of Research and Development*, vol. 52, no. 1.2, pp. 199–220, Jan. 2008. DOI: 10.1147/rd.521.0199 (see p. 21)

[86] *IEEE 802.3ad-2000 - IEEE standard for information technology - local and metropolitan area networks - part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications-aggregation of multiple link segments*, IEEE Computer Society, Jul. 2000 (see p. 72)

[87] *IEEE 802.3x-1997 - IEEE standards for local and metropolitan area networks: Supplements to carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications - specification for 802.3 full duplex operation and physical layer specification for 100 Mb/s operation on two pairs of category 3 or better balanced twisted pair cable (100BASE-T2)*, IEEE Computer Society, Jul. 2000 (see p. 61)

[88] *IEEE standard for local and metropolitan area networks–media access control (MAC) bridges and virtual bridged local area networks - amendment 17: Priority-based flow control, 802.1Qbb*, IEEE Computer Society, 2011 (see pp. 61, 72, 157)

[89] *IEEE standard for local and metropolitan area networks–virtual bridged local area networks - amendment: 10: Congestion notification, 802.1Qau*, IEEE Computer Society, 2010 (see pp. 37, 158, 166)

[90] *IEEE standard for local and metropolitan area networks: Virtual bridged local area networks - amendment 8: Shortest path bridging, 802.1aq*, IEEE Computer Society, Mar. 2012 (see p. 157)

[91] IEEE Standards Association, *Assigned Ethertype values list*, accessed: 2015-02-17, Mar. 2005. [Online]. Available: `http://standards.ieee.org/develop/regauth/ethertype/eth.txt` (see p. 71)

[92] Infiniband Trade Association, *Supplement to InfiniBand architecture specification volume 1 release 1.2.1. annex a17: RoCEv2*, accessed: 2015-01-20, Sep. 2, 2014. [Online]. Available: `https://cw.infinibandta.org/document/dl/7781` (see pp. 64, 158)

[93] N. Jain, A. Bhatele, X. Ni, N. J. Wright, and L. V. Kale, "Maximizing throughput on a Dragonfly network," in *SC'14: International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, Nov. 2014. DOI: `10.1109/sc.2014.33` (see pp. 53, 55)

[94] N. Jiang, D. U. Becker, G. Michelogiannakis, and W. J. Dally, "Network congestion avoidance through speculative reservation," in *IEEE International Symposium on High-Performance Computer Architecture*, Feb. 2012, pp. 1–12 (see p. 160)

[95] N. Jiang, L. Dennison, and W. J. Dally, "Network endpoint congestion control for fine-grained communication," in *SC'15: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–12 (see p. 160)

[96] N. Jiang, J. Kim, and W. J. Dally, "Indirect adaptive routing on large scale interconnection networks," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, p. 220, Jun. 2009. DOI: `10.1145/1555815.1555783` (see pp. ii, 32, 33, 61, 76, 85, 105, 108, 129, 141, 158, 159)

[97] M. Jurczyk and T. Schwederski, "Phenomenon of higher order head-of-line blocking in multistage interconnection networks under nonuniform traffic patterns," *IEICE Transactions on Information and Systems*, vol. 79, pp. 1124–1129, 1996 (see p. 80)

[98] R. Kahn and W. Crowther, "Flow control in a resource-sharing computer network," *IEEE Transactions on Communications*, vol. 20, no. 3, pp. 539–546, Jun. 1972. DOI: `10.1109/tcom.1972.1091202` (see p. 34)

[99] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, pp. 51–62, Mar. 2007, ISSN: 0146-4833. DOI: `10.1145/1232919.1232925` (see pp. 76, 158)

[100] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *9th ACM SIGCOMM Conference on Internet Measurement Conference*, Chicago, Illinois, USA: ACM, 2009, pp. 202–208, ISBN: 978-1-60558-771-4 (see p. 62)

[101] K. Kannan and S. Banerjee, "Compact TCAM: Flow entry compaction in TCAM for power aware SDN," English, in *Distributed Computing and Networking*, ser. Lecture Notes in Computer Science, D. Frey, M. Raynal, S. Sarkar, R. Shyamasundar, and P. Sinha, Eds., vol. 7730, Springer Berlin Heidelberg, 2013, pp. 439–444, ISBN: 978-3-642-35667-4. DOI: 10.1007/978-3-642-35668-1_32 (see p. 82)

[102] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Trans. Communications*, vol. 35, pp. 1347–1356, 1987 (see pp. 14, 15)

[103] G. Kathareios, C. Minkenberg, B. Prisacari, G. Rodriguez, and T. Hoefler, "Cost-effective diameter-two topologies: Analysis and evaluation," in *SC'15: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2015, pp. 1–11. DOI: 10.1145/2807591.2807652 (see p. 161)

[104] D. J. Kerbyson and K. J. Barker, "Analyzing the performance bottlenecks of the POWER7-IH network," in *2011 IEEE International Conference on Cluster Computing*, IEEE, Sep. 2011. DOI: 10.1109/cluster.2011.35 (see p. 55)

[105] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks (1976)*, vol. 3, no. 4, pp. 267–286, Sep. 1979. DOI: 10.1016/0376-5075(79)90032-1 (see p. 34)

[106] C. Kim, M. Caesar, and J. Rexford, "SEATTLE: A scalable ethernet architecture for large enterprises," *ACM Trans. Comput. Syst.*, vol. 29, no. 1, 1:1–1:35, 2011, ISSN: 0734-2071. DOI: 10.1145/1925109.1925110 (see p. 71)

[107] J. Kim, W. Dally, B. Towles, and A. Gupta, "Microarchitecture of a high-radix router," in *32nd International Symposium on Computer Architecture (ISCA'05)*, IEEE, 2005. DOI: 10.1109/isca.2005.35 (see p. 13)

[108] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable Dragonfly topology," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, pp. 77–88, Jun. 2008. DOI: 10.1145/1394608.1382129 (see pp. ii, iv, 8, 21, 22, 24, 26, 30, 32, 39, 52, 57, 61, 66, 74, 76, 105, 133, 158, 160)

[109] J. Kim, W. J. Dally, and D. Abts, "Adaptive routing in high-radix Clos network," in *SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, 2006. DOI: 10.1109/SC.2006.10 (see p. 158)

[110] ——, "Flattened Butterfly: A cost-efficient topology for high-radix networks," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, p. 126, Jun. 2007. DOI: 10.1145/1273440.1250679 (see pp. iv, 8, 21, 22, 39, 66, 74, 158)

[111] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, K. Yelick, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Keckler, D. Klein, P. Kogge, R. S. Williams, and K. Yelick, *ExaScale computing study: Technology challenges in achieving exascale systems*, Sep. 2008 (see p. 84)

[112] P. Kogge and J. Shalf, "Exascale computing trends: Adjusting to the new normal for computer architecture," *Computing in Science & Engineering*, vol. 15, no. 6, pp. 16–26, Nov. 2013. DOI: 10.1109/mcse.2013.95 (see pp. i, 1)

[113] T. Kohonen, *Content-Addressable Memories, 2nd ed.* Springer Berlin Heidelberg, 1987, ISBN: 9783540176251. DOI: 10.1007/978-3-642-83056-3 (see p. 17)

[114] D. Kreutz, F. M. V. Ramos, P. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *CoRR*, vol. abs/1406.0440, 2014 (see p. 67)

[115] R. Krishnan, L. Yong, A. Ghanwani, N. So, and B. Khasnabish, "Mechanisms for optimizing link aggregation group (LAG) and equal-cost multipath (ECMP) component link utilization in networks (RFC 7424)," Internet Engineering Task Force (IETF), RFC 7424, Jan. 2015, ISSN: 2070-1721, accessed: 2015-05-14. [Online]. Available: https://tools.ietf.org/html/rfc7424 (see p. 158)

[116] H. T. Kung, T. Blackwell, and A. Chapman, "Credit-based flow control for ATM networks," in *Proceedings of the conference on Communications architectures, protocols and applications - SIGCOMM'94*, ACM Press, Oct. 1994. DOI: 10.1145/190314.190324 (see p. 36)

[117] J. Labarta, S. Girona, and T. Cortes, "Analyzing scheduling policies using Dimemas," *Parallel Computing*, vol. 23, no. 1-2, pp. 23–34, Apr. 1997. DOI: 10.1016/s0167-8191(96)00094-4 (see p. 51)

[118] J. Laudon and D. Lenoski, "The SGI origin," in *Proceedings of the 24th annual international symposium on Computer architecture - ISCA'97*, ACM Press, 1997. DOI: 10.1145/264107.264206 (see p. 21)

[119] Lawrence Livermore National Laboratory, *Sierra supercomputer*, accessed: 2020-08-26. [Online]. Available: https://computing.llnl.gov/computers/sierra (see pp. 21, 157)

[120] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, Oct. 1985. DOI: 10.1109/tc.1985.6312192 (see p. 21)

[121] S.-Y. Li, "Theory of periodic contention and its application to packet switching," in *IEEE INFOCOM'88, Seventh Annual Joint Conference of the IEEE Computer and Communcations Societies. Networks: Evolution or Revolution?*, IEEE, 1988. DOI: 10.1109/infcom.1988.12933 (see p. 15)

[122] Los Alamos National Laboratory, *Trinity supercomputer*, accessed: 2020-08-26. [Online]. Available: https://www.lanl.gov/projects/trinity/ (see p. 22)

[123] C. Macapuna, C. Rothenberg, and M. Magalhães, "In-packet Bloom filter based data center networking with distributed OpenFlow controllers," in *IEEE GLOBECOM Workshops (GC Wkshps)*, Dec. 2010, pp. 584–588. DOI: 10.1109/GLOCOMW.2010.5700387 (see p. 157)

[124] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks (RFC 7348)," Independent Submission, RFC 7348, Aug. 2014, ISSN: 2070-1721, accessed: 2015-01-16. [Online]. Available: https://tools.ietf.org/html/rfc7348 (see pp. 65, 157)

[125] A. McAuley and P. Francis, "Fast routing table lookup using CAMs," in *IEEE INFOCOM'93 The Conference on Computer Communications, Proceedings*, IEEE Comnput. Soc. Press, 1993. DOI: 10.1109/infcom.1993.253403 (see p. 17)

[126] N. McDonald, M. Isaev, A. Flores, A. Davis, and J. Kim, "Practical and efficient incremental adaptive routing for HyperX networks," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ACM, Nov. 2019. DOI: 10.1145/3295500.3356151 (see p. 26)

[127] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, Apr. 1999. DOI: 10.1109/90.769767 (see p. 18)

[128] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, "Tiny Tera: A packet switch core," *IEEE Micro*, vol. 17, no. 1, pp. 26–33, 1997. DOI: 10.1109/40.566194 (see pp. 14, 15)

[129] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, Mar. 2008. DOI: 10.1145/1355734.1355746 (see pp. 41, 61)

[130] Mellanox Inc., "RoCE in the data center," Mellanox, Tech. Rep., Oct. 2014 (see pp. 64, 158)

[131] ——, "InfiniBand port counters," Mellanox, Tech. Rep., Apr. 9, 2019, accessed: 2020-07-30. [Online]. Available: https://community.mellanox.com/s/article/infiniband-port-counters (see pp. vii, 131, 158)

[132] ——, "RoCE v2 consideration," Mellanox, Tech. Rep., Apr. 2019, accessed: 2020-08-23. [Online]. Available: https://community.mellanox.com/s/article/roce-v2-considerations (see p. 64)

[133] L. Mhamdi, "PBC: A partially buffered crossbar packet switch," *IEEE Transactions on Computers*, vol. 58, no. 11, pp. 1568–1581, Nov. 2009. DOI: 10.1109/tc.2009.65 (see p. 14)

[134] G. Michelogiannakis, N. Jiang, D. Becker, and W. J. Dally, "Channel reservation protocol for over-subscribed channels and destinations," in *SC'13: Intl Conf. High Performance Computing, Networking, Storage and Analysis*, Nov. 2013, pp. 1–12 (see p. 160)

[135] M. Miller and J. Sirán, "Moore graphs and beyond: A survey of the degree/diameter problem," *The Electronic Journal of Combinatorics*, vol. 1000, May 2013. DOI: 10.37236/35 (see pp. iv, 8)

[136] C. Minkenberg, M. Gusat, and G. Rodriguez, "Adaptive routing in data center bridges," in *17th IEEE Symposium on High Performance Interconnects (HOTI)*, 2009, pp. 33–41. DOI: 10.1109/HOTI.2009.14 (see p. 159)

[137] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee, "DevoFlow: Cost-effective flow management for high performance enterprise networks," in *SIGCOMM Workshop on Hot Topics in Networks*, 2010, ISBN: 978-1-4503-0409-2 (see pp. 62, 76, 158)

[138] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The Alpha 21364 network architecture," *IEEE Micro*, vol. 22, no. 1, pp. 26–35, 2002. DOI: 10.1109/40.988687 (see p. 21)

[139] H. M. Mulder, "Interval-regular graphs," *Discrete Mathematics*, vol. 41, no. 3, pp. 253–269, 1982. DOI: 10.1016/0012-365x(82)90021-8 (see p. 21)

[140] M. Nabeshima, "Performance evaluation of a combined input- and crosspoint-queued switch," *IEICE Transactions on Communications*, pp. 737–741, Jan. 2000 (see p. 14)

[141] NEC Corporation, "NEC ProgrammableFlow: An open and programmable network fabric for datacenters and the cloud," NEC Corporation, Tech. Rep., 2012 (see p. 160)

[142] F. D. Neeser, N. I. Chrysos, R. Clauberg, D. Crisan, M. Gusat, C. Minkenberg, K. M. Valk, and C. Basso, "Occupancy sampling for terabit CEE switches," in *20th Annual Symposium on High-Performance Interconnects*, 2012, pp. 64–71. DOI: 10.1109/HOTI.2012.14sci (see pp. 37, 83, 91)

[143] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM Conference on Data Communication*, ser. SIGCOMM '09, Barcelona, Spain: ACM, 2009, pp. 39–50, ISBN: 978-1-60558-594-9 (see pp. 65, 71, 157)

[144] Open Compute Project Community, *Open compute project network specifications and designs*, accessed: 2015-12-17, 2015. [Online]. Available: http://www.opencompute.org/wiki/Networking/SpecsAndDesigns (see p. 61)

[145] Open Networking Foundation, *OpenFlow switch specification version 1.0.0, ONF TS-001*, accessed: 2015-01-13, Dec. 2009. [Online]. Available: http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf (see pp. 41, 71)

[146] ——, *OpenFlow switch specification version 1.5.1, ONF TS-025*, accessed: 2020-08-22, Mar. 2015. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf (see p. 42)

[147] C. Ozveren, R. Simcoe, and G. Varghese, "Reliable and efficient hop-by-hop flow control," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 4, pp. 642–650, May 1995. DOI: 10.1109/49.382155 (see p. 36)

[148] M. Parker, S. Scott, A. Cheng, and J. Kim, "Progressive adaptive routing in a Dragonfly processor interconnect network," US 9,137,143 B2, Sep. 15, 2015 (see p. 161)

[149] C. Partridge, P. Carvey, E. Burgess, I. Castineyra, T. Clarke, L. Graham, M. Hathaway, P. Herman, A. King, S. Kohlami, T. Ma, J. Mcallen, T. Mendez, W. Milliken, R. Osterlind, R. Pettyjohn, J. Rokosz, J. Seeger, M. Sollins, S. Storch, B. Tober, G. Troxel, D. Waitzman, and S. Winterble, "A fifty gigabit per second IP router," *IEEE/ACM Transuctions on Networking*, 1998 (see p. 15)

[150] I. Pérez, E. Vallejo, and R. Beivide, "SMART++: Reducing cost and improving efficiency of multi-hop bypass in NoC routers," in *Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip - NOCS'19*, ACM Press, 2019. DOI: 10.1145/3313231.3352364 (see p. 35)

[151] R. Perlman, "RBridges: Transparent routing," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 2, 2004, 1211–1218 vol.2. DOI: 10.1109/INFCOM.2004.1357007 (see pp. 64, 157)

[152] G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, "Solving hot spot contention using InfiniBand architecture congestion control," in *High Performance Interconnects for Distributed Computing Workshop (HPI-DC)*, 2005 (see pp. 37, 158)

[153] G. F. Pfister, "An introduction to the InfiniBand architecture," *High Performance Mass Storage and Parallel I/O*, vol. 42, pp. 617–632, 2001 (see pp. 7, 64, 157)

[154] M. Ponce, R. van Zon, S. Northrup, D. Gruner, J. Chen, F. Ertinaz, A. Fedoseev, L. Groer, F. Mao, B. C. Mundim, M. Nolta, J. Pinto, M. Saldarriaga, V. Slavnic, E. Spence, C.-H. Yu, and W. R. Peltier, "Deploying a top-100 supercomputer for large parallel workloads," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning) - PEARC'19*, ACM, Jul. 2019. DOI: 10.1145/3332186.3332195 (see p. 22)

[155] B. Prabhakar and N. McKeown, "On the speedup required for combined input- and output-queued switching," *Automatica*, vol. 35, no. 12, pp. 1909–1920, Dec. 1999. DOI: 10.1016/s0005-1098(99)00129-6 (see pp. 14, 15)

[156] M. S. Rahman, S. Bhowmik, Y. Ryasnianskiy, X. Yuan, and M. Lang, "Topology-custom UGAL routing on Dragonfly," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ACM, Nov. 2019. DOI: 10.1145/3295500.3356208 (see pp. 105, 161)

[157] N. Rajovic, A. Rico, F. Mantovani, D. Ruiz, J. O. Vilarrubi, C. Gomez, L. Backes, D. Nieto, H. Servat, X. Martorell, J. Labarta, E. Ayguade, C. Adeniyi-Jones, S. Derradji, H. Gloaguen, P. Lanucara, N. Sanna, J.-F. Mehaut, K. Pouget, B. Videau, E. Boyer, M. Allalen, A. Auweter, D. Brayford, D. Tafani, V. Weinberg, D. Brommel, R. Halver, J. H. Meinke, R. Beivide, M. Benito, E. Vallejo, M. Valero, and A. Ramirez, "The Mont-Blanc prototype: An alternative approach for HPC systems," in *SC'16: International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, Nov. 2016. DOI: 10.1109/sc.2016.37 (see pp. iv, 8, 9, 157)

[158] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ECN) to IP (RFC 3168)," Network Working Group, RFC 3168, Sep. 2001, accessed: 2015-02-3. [Online]. Available: https://tools.ietf.org/html/rfc3168 (see pp. 37, 158)

[159] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca, "Planck: Millisecond-scale monitoring and control for commodity networks," in *ACM Conference on SIGCOMM*, Chicago, Illinois, USA: ACM, 2014, pp. 407–418, ISBN: 978-1-4503-2836-4 (see pp. 63, 76, 158)

[160] R. Rojas-Cessa, E. Oki, and H. J. Chao, "CIXOB-k: Combined input-crosspoint-output buffered packet switch," in *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*, IEEE, 2001. DOI: 10.1109/glocom.2001.966256 (see p. 14)

[161] K. Saravanan, P. Carpenter, and A. Ramirez, "Power/performance evaluation of energy efficient ethernet (EEE) for high performance computing," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, 2013, pp. 205–214. DOI: 10.1109/ISPASS.2013.6557171 (see p. 160)

[162] R. Schaller, "Moore's law: Past, present and future," *IEEE Spectrum*, vol. 34, no. 6, pp. 52–59, Jun. 1997. DOI: 10.1109/6.591665 (see pp. i, 1)

[163] M. Scott, A. Moore, and J. Crowcroft, "Addressing the scalability of Ethernet with MOOSE," in *Proc. DC CAVES Workshop*, 2009 (see pp. 65, 71, 157)

[164] S. Scott, D. Abts, J. Kim, and W. Dally, "The BlackWidow high-radix clos network," in *33rd International Symposium on Computer Architecture (ISCA'06)*, IEEE, 2006. DOI: 10.1109/isca.2006.40 (see pp. 13, 14, 158)

[165] S. L. Scott and G. M. Thorson, "The Cray T3E network: Adaptive routing in a high performance 3D torus," 1996 (see p. 21)

[166] D. D. Sensi, S. D. Girolamo, K. H. McMahon, D. Roweth, and T. Hoefler, "An in-depth analysis of the Slingshot interconnect," *CoRR*, vol. abs/2008.08886, 2020, Accessed: 2020-08-26. arXiv: 2008.08886. [Online]. Available: https://arxiv.org/abs/2008.08886 (see pp. 6, 22, 157, 158)

[167] D. Serpanos, M. Katevenis, and E. Spyridakis, "ATLAS I: Building block for ATM networks with credit-based flow control," in *The Fourth IEEE Workshop on High-Performance Communication Systems*, IEEE, Jun. 1997. DOI: 10.1109/hpcs.1997.864038 (see p. 36)

[168] D. Serpanos and T. Wolf, *Architecture of network systems*, 1st. San Francisco, CA, USA: Morgan Kaufmann, 2011, ISBN: 9780080922829. DOI: 10.5555/1995302 (see p. 13)

[169] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in *ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS'13, Berlin, Germany: ACM, 2013, pp. 413–424, ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516684 (see p. 160)

[170] A. Shpiner, Z. Haramaty, S. Eliad, V. Zdornov, B. Gafni, and E. Zahavi, "Dragonfly+: Low cost topology for scaling datacenters," in *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, IEEE, Feb. 2017. DOI: 10.1109/hipineb.2017.11 (see pp. 21, 158)

[171] H. J. Siegel, *Interconnection networks for large-scale parallel processing : theory and case studies (2nd Ed.)* New York: McGraw-Hill, 1990, ISBN: 0070575614 (see p. 19)

[172] A. Singh, "Load-balanced routing in interconnection networks," accessed: 2014-7-7, Ph.D. dissertation, Stanford University, 2005. [Online]. Available: http://cva.stanford.edu/publications/2005/thesis_arjuns.pdf (see pp. ii, 31, 103, 129, 159)

[173] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, 2012, pp. 225–238 (see p. 161)

[174] B. Stephens, A. Cox, A. Singla, J. Carter, C. Dixon, and W. Felter, "Practical DCB for improved data center networks," in *INFOCOM, 2014 Proceedings IEEE*, 2014, pp. 1824–1832. DOI: 10.1109/INFOCOM.2014.6848121 (see p. 160)

[175] C. B. Stunkel, R. L. Graham, G. Shainer, M. Kagan, S. S. Sharkawi, B. Rosenburg, and G. A. Chochia, "The high-speed networks of the Summit and Sierra supercomputers," *IBM Journal of Research and Development*, vol. 64, no. 3/4, 3:1–3:10, 2020 (see pp. 21, 157)

[176] Y. Sun and M. S. Kim, "A hybrid approach to CAM-based longest prefix matching for IP route lookup," in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, IEEE, Dec. 2010. DOI: 10.1109/glocom.2010.5683639 (see p. 17)

[177] Y. Tamir and G. L. Frazier, "High-performance multi-queue buffers for VLSI communications switches," *ACM SIGARCH Computer Architecture News*, vol. 16, no. 2, pp. 343–354, May 1988. DOI: 10.1145/633625.52439 (see p. 15)

[178] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in MPICH," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, Feb. 2005. DOI: 10.1177/1094342005051521 (see p. 54)

[179] *Top500 supercomputer ranking*, accessed: 2020-08-10, 2020. [Online]. Available: http://www.top500.org (see pp. iii, 61)

[180] S. Toueg and K. Steiglitz, "Some complexity results in the design of deadlock-free packet switching networks," *SIAM Journal on Computing*, vol. 10, no. 4, pp. 702–712, Nov. 1981. DOI: 10.1137/0210053 (see pp. ii, 38)

[181] B. Towles and W. Dally, "Worst-case traffic for oblivious routing functions," *IEEE Computer Architecture Letters*, vol. 1, no. 1, pp. 4–4, Jan. 2002. DOI: 10.1109/l-ca.2002.12 (see p. 30)

[182] W. H. Tranter, D. P. Taylor, R. E. Ziemer, N. F. Maxemchuk, and J. W. Mark, "Input versus output queueing on a SpaceDivision packet switch," in *The Best of the Best: Fifty Years of Communications and Networking Research*. 2007, pp. 561–570 (see p. 15)

[183] U.S Department of Energy (DOE) - Office of Science, *Exascale computing project*, accessed: 2015-06-18. [Online]. Available: https://www.exascaleproject.org (see p. 3)

[184] ——, *Summit Oak Ridge national laboratory's 200 petaflop supercomputer*, accessed: 2020-08-13. [Online]. Available: https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit (see pp. 21, 61, 157)

[185] M. Valerio, L. Moser, and P. Melliar-Smith, "Recursively scalable fat-trees as interconnection networks," in *Proceeding of 13th IEEE Annual International Phoenix Conference on Computers and Communications*, IEEE, 1994. DOI: 10.1109/pccc.1994.504091 (see p. 161)

[186] L. G. Valiant, "A scheme for fast parallel communication," *SIAM Journal on Computing*, vol. 11, no. 2, pp. 350–361, May 1982. DOI: 10.1137/0211027 (see pp. ii, 8, 29)

[187] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *Proceedings of the thirteenth annual ACM symposium on Theory of computing - STOC'81*, ACM Press, 1981. DOI: 10.1145/800076.802479 (see pp. ii, 8, 29, 103, 161)

[188] J. Wade and C. Sodini, "Dynamic cross-coupled bitline content addressable memory cell for high density arrays," in *1985 International Electron Devices Meeting*, IRE, 1985. DOI: 10.1109/iedm.1985.190952 (see p. 17)

[189] Y. T. Wang and B. Sengupta, "Performance analysis of a feedback congestion control policy under non-negligible propagation delay," *ACM SIGCOMM Computer Communication Review*, vol. 21, no. 4, pp. 149–157, Aug. 1991. DOI: 10.1145/115994.116007 (see p. 35)

[190] Y. Wang, G. Lu, and X. Li, "A study of internet packet reordering," in *Information Networking. Networking Technologies for Broadband and Mobile Networks*, ser. Lecture Notes in Computer Science, vol. 3090, Springer Berlin Heidelberg, 2004, pp. 350–359, ISBN: 978-3-540-23034-2. DOI: 10.1007/978-3-540-25978-7_36 (see p. 158)

[191] P. Wijetunga, "High-performance crossbar design for system-on-chip," in *The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications, 2003. Proceedings.*, IEEE Comput. Soc, 2003. DOI: 10.1109/iwsoc.2003.1213023 (see p. 14)

[192] J. Won, G. Kim, J. Kim, T. Jiang, M. Parker, and S. Scott, "Overcoming far-end congestion in large-scale networks," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2015, pp. 415–427 (see pp. 79, 125, 159)

[193] P. Yébenes, J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, and T. Hoefler, "Improving non-minimal and adaptive routing algorithms in Slim Fly networks," in *2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI)*, Aug. 2017, pp. 1–8. DOI: 10.1109/HOTI.2017.11 (see pp. 106, 161)

[194] J. Y. Yen, "Finding the K shortest loopless paths in a network," *Management Science*, vol. 17, no. 11, pp. 712–716, 1971, ISSN: 00251909, 15265501 (see p. 161)

[195] E. Zahavi, I. Keslassy, and A. Kolodny, "Distributed adaptive routing for big-data applications running on data center networks," in *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems - ANCS'12*, ACM Press, 2012. DOI: 10.1145/2396556.2396578 (see pp. 157, 158)

[196] H. Zhai, F. Hu, R. Perlman, D. E. 3rd, and O. Stokes, "Transparent interconnection of lots of links (TRILL): End station address distribution information (ESADI) protocol (RFC 7357)," Internet Engineering Task Force (IETF), RFC 7357, Sep. 2014, ISSN: 2070-1721, accessed: 2015-02-6. [Online]. Available: https://tools.ietf.org/html/rfc7357 (see pp. 65, 159)

[197] Y. Zhang, O. Tuncer, F. Kaplan, K. Olcoz, V. J. Leung, and A. K. Coskun, "Levelspread: A new job allocation policy for Dragonfly networks," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, May 2018. DOI: 10.1109/ipdps.2018.00121 (see p. 53)